1968

An investigation of the use of multiple pivot selection rules to solve integer programming problems

Bruce Allen Weingartner
Lehigh University

AN INVESTIGATION OF THE USE OF MULTIPLE PIVOT SELECTION RULES TO SOLVE INTEGER PROGRAMMING PROBLEMS

by
Bruce Allen Weingartner

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in Industrial Engineering

Lehigh University
1968

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 14, 1968
Date

Gary E. Whitehouse
Professor in Charge

G. E. Kane
Head of the Department of
Industrial Engineering

ACKNOWLEDGEMENTS

The author wishes to thank his thesis advisor, Dr. G. E. Whitehouse, for his guidance and interest in the work presented here. Thanks are also due Professors J. M. Carroll and G. E. Kane for serving as members of the advisory committee.

I particularly wish to acknowledge the advice and assistance of Dr. J. H. Schmaltz of the Western Electric Co., Inc., Engineering Research Center. His suggestions led to the learning technique advanced in this paper. Thanks are also due to Dr. R. P. Thayer and R. F. Storer of the Western Electric Co., Inc., Engineering Research Center, for their assistance in the statistical portions of this paper. I also wish to thank Miss P. A. Renzo for typing this paper.

Last, but by no means least, I wish to thank my wife, Angela, and son, Kenneth. Their patience and encouragement contributed immeasurably to the completion of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

A frequently encountered problem in industrial applications is the generalized transportation problem with the added restriction that the solution must consist of integer variables. Several existing all-integer integer programming pivot selection rules are investigated and compared with an algorithm advanced by the author. This algorithm, known as a learning technique, decides which of the pivot selection rules it should use as it progresses through the problem. This decision is based on which rule is progressing the objective value the best. Twenty-five randomly selected problems are used to statistically test the various methods to see which performs the best. The results of this experiment indicate that the learning algorithm is significantly better than the individual pivot selection rules tested, on problems of this structure.

I INTRODUCTION

A frequently encountered problem in industrial applications is that of allocating limited resources among competing activities in an optimal manner. Generally, to solve such a problem, a mathematical model is first constructed. If all the functions of the model are linear, then linear programming techniques[6,20] may be applied to find the optimal solution.

A general statement of a linear programming problem is the following:

Find $x_1, x_2, \ldots, x_n$ in order to

$$(1) \quad \text{maximize } f(x) = \sum_{j=1}^{n} c_j x_j$$

subject to the restrictions

$$(2) \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1,2,\ldots,m$$

and $$(3) \quad x_j \geq 0 \qquad j = 1,2,\ldots,n$$

where the a_{ij}, b_i and c_j are given constants.

The function to be maximized (1) is referred to as the objective function, the restrictions (2) and (3) are referred to as constraints, while the variables to be solved for (x_j) may be thought of as decision variables. Thus, in this case, we have n activities competing for m resources. In the final solution, x_j will represent the level of the j^{th} activity. For example, if the j^{th} activity is the manufacture of a certain product, x_j would represent the number of units

of the product to be manufactured. In many practical problems the decision variables make sense only if they are integer valued. For example, if we desire to assign men, machines, or vehicles to a set of activities it is clear that these variables must generally be integer valued. This additional restriction places the problem in the realm of integer linear programming (or, more simply, integer programming).

Chapter II of this thesis will present the structure of a specific integer programming problem, while the ensuing chapters will describe and evaluate specific methods for solving problems of this structure. Before getting into this problem, however, it is best to first review the field of integer programming in general.

The determination of integer solutions of algebraic equations is one of the most difficult problems in number theory[9]. Such famous mathematicians as Pythagoras and Diophantus were concerned with this problem more than 2300 years ago. More recently P. Fermat, L. Euler and J. Lagrange, among others, have also considered this problem. However, the first systematic approach to the integer programming problem as such was not accomplished until 1958 by R. E. Gomory[14]. It is interesting to note that Beale[4] states that until Gomory's breakthrough many people thought that a general method of solving integer programming problems was self-evidently impossible. In the decade since this first development, a variety of algorithms for solving integer programming problems have been formulated.

Integer programming problems may be broadly categorized into two

groups: "mixed" integer problems, and "pure" integer problems. Mixed integer problems are problems in which only some of the decision variables are restricted to be integer valued, whereas in pure integer problems all of the variables must be integer valued. An important subset of the pure integer problem is one where the variables are restricted to be either 0 or 1, and several algorithms have been developed specifically to handle this case.

Generally, one of the first impulses in attempting to solve integer programming problems is to solve the problem as a linear programming problem (i.e., neglecting the constraint that the variables must be integer) and then rounding the solution obtained into integer values. However, such an approach leads to several difficulties. First, it is difficult to determine which variables should be rounded up and which down. Second, after rounding is performed some of the constraints may be violated and if not, the rounded solution may still not be the optimal integer solution. To illustrate these points, consider the following problem:

$$\text{Maximize } f(x) = 4x_1 + 7x_2$$

$$\text{subject to } x_1 + 2x_2 \leq 8$$

$$2x_1 - x_2 \leq 5$$

Figure 1 shows the graphical solution to this problem. The linear programming solution is at $x_1 = 3.6$, $x_2 = 2.2$, $f(x) = 29.8$. Observe that if we round this answer off to $x_1 = 4$, $x_2 = 2$ we are outside the feasible region. Similarly, the points (4,3) and (3,3) also lie outside the region. Therefore we could round the solution to $x_1 = 3$,

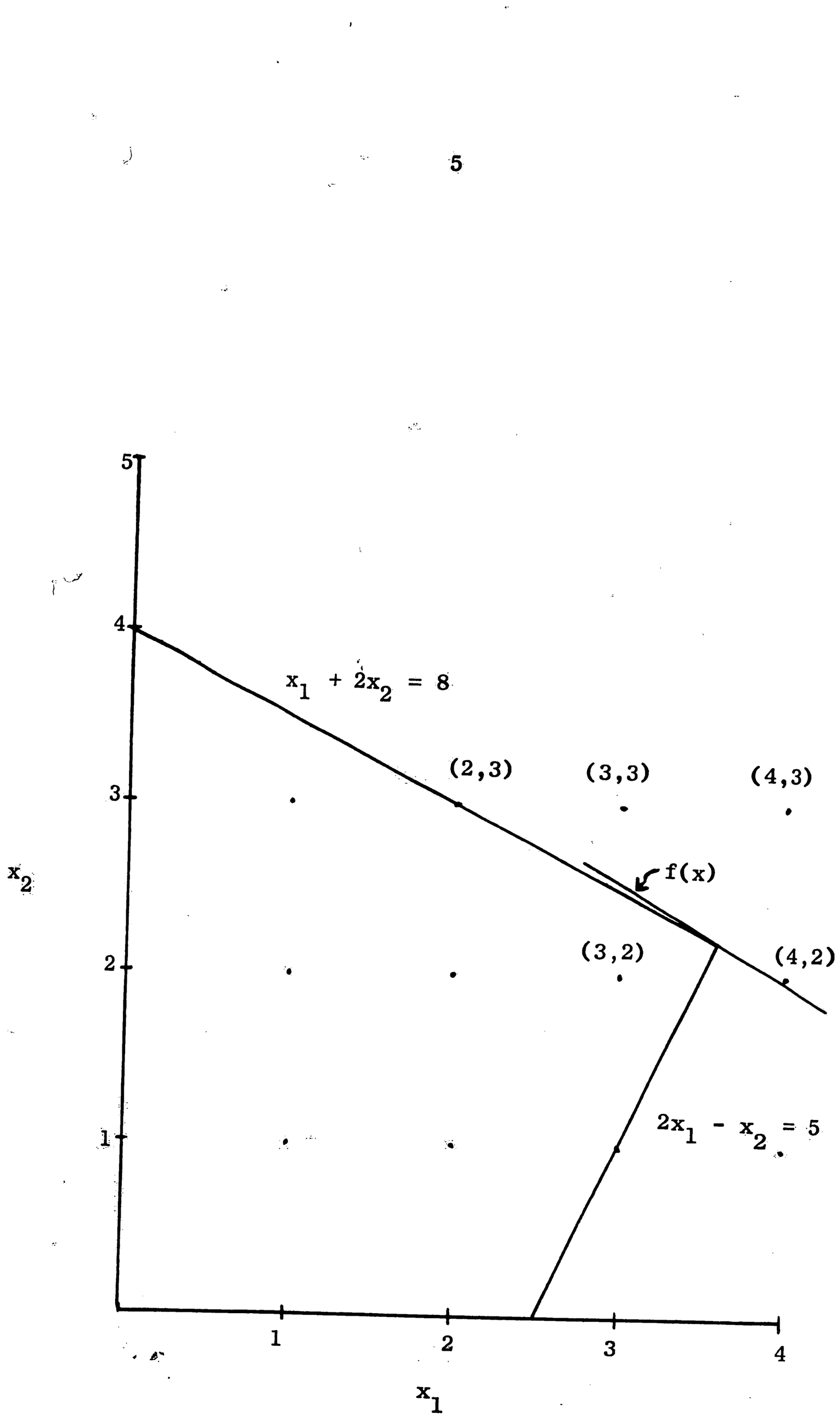


FIGURE 1 GRAPHICAL REPRESENTATION
OF A SMALL PROBLEM

$x_2 = 2$, $f(x) = 26$, which is the feasible integer point nearest the linear programming solution. However, further analysis shows that the optimal integer solution lies at $x_1 = 2$, $x_2 = 3$, $f(x) = 29$. While the optimal solution can easily be found for this problem through trial and error, the difficulty is greatly compounded for problems involving many variables. Despite these difficulties R. E. Gomory[17] has published a paper dealing with a rounding algorithm. The procedure presented is basically a dynamic programming process for rounding the linear programming solution. Little computational experience has been reported using this method.

The most famous and probably most commonly used integer programming algorithms are the two cutting plane techniques of Gomory. The first of these algorithms, introduced in 1958, is the method of integer forms or fractional method[14]. Basically this algorithm solves the problem using linear programming and then checks to see if the solution satisfies the integer constraints. If the solution is not all integers an additional constraint is added to the problem. The additional constraint is generated from one of the fractional variables in the solution. The effect of this constraint is to cut into the feasible region in such a manner that the present non-integer solution is eliminated from further consideration but no feasible integer solutions are eliminated. The problem is then re-solved and the process continued until an all integer solution is obtained. While it has been proved that this method will guarantee an integer solution in a finite number of steps, several computational difficulties have been encountered when using this technique. The major difficulties are

that (1) because the method deals with fractions, computer round-off errors can occur, (2) the size of the problem may be drastically increased due to the number of additional constraints added and retained as the method progresses towards a solution, and (3) the rate of convergence to an integer solution may be intolerably slow and highly erratic.

In 1960, Gomory introduced his second algorithm, known as the all-integer algorithm[13]. This method has the advantages of eliminating the round-off problem and of being able to discard each constraint after having used it, thus preventing the size of the problem from increasing. However, the initial size of the problem may be larger than in the method discussed above due to the format used by this method. The rate of convergence of this method may also be slow and erratic and an additional disadvantage is that all intermediate solutions are infeasible. This is because the dual simplex method is used, which allows negative decision variables to exist in the intermediate solutions. Thus,if the process is terminated before the optimal integer solution has been found, the intermediate solution obtained is of little or no value. This method, which is used in this thesis, is explained in detail in Chapter III.

In general, computational experience using these cutting plane techniques can be described as somewhat erratic and unpredictable. It is not uncommon to find problems, including some very small ones, ~~which fail~~ to converge to a solution in a tolerable amount of time. Results of computational experience can be found in reference such as

Balinski[2], Haldi and Isaacson[19], Schmaltz[28], Srinivasan[29], Story and Wagner[30], and Trauth and Woolsey[31].

The above cutting plane techniques are for pure integer problems. However, Gomory[15] has extended the method of integer forms to deal with mixed integer problems as well.

A second major approach to the solution of integer programming problems is the use of the Branch and Bound method. One such technique was proposed by Land and Doig in 1960[22]. This method consists of setting up a tree of linear programming problems. First the linear programming solution to the original problem is obtained. If this solution does not satisfy the integer constraints, the branching process is begun with the value of the objective function serving as an upper bound on the value of the optimal integer solution. Each branch of the tree forces a specific variable to take on an integer value within its possible range. The problem is re-solved at each node of the tree and a new bound on the objective function is obtained. The process is continued until a solution is obtained that satisfies the integer constraints and which can be shown to be optimal from the bounds obtained. This method is sometimes referred to as a shifted hyperplane technique since its effect is to shift the hyperplane defining the objective function so it cuts into the feasible region. The Land and Doig method may be applied to either pure or mixed integer problems. Its main application seems to be to mixed problems where only a small number of the variables are required to be integer valued.

An obvious approach to solving integer programming problems is complete enumeration. That is, find all the feasible combinations of integer solutions and then choose the best solution by comparison. Clearly such an approach is impractical for large, complex problems, since an astronomical number of combinations would have to be explored. Techniques have been developed which can exclude many of the possible combinations; in fact, the Land and Doig method discussed above may be considered as a form of enumerative method. In general these techniques deal with partitioning the set of all feasible integer solutions into subsets and establishing bounds on the value of the objective function of the solutions contained in each subset. Based on the values of the bounds certain subsets can by systematically eliminated and others partitioned further until an optimal solution is obtained. In the worst possible case, this method would require complete enumeration of all solutions. Much work is presently being done on how to best partition the set of feasible solutions and how to establish the most discriminating bounds. The survey papers of Glover[12], and Lawler and Woods[23] are recommended for further details on this subject.

An interesting algorithm of the enumerative type was published by Balas[1] in 1965. This algorithm deals with the special class of problems in which the variables are restricted to take on only the values of zero or one. A major feature of this algorithm is that it is completely additive, so that round-off errors are avoided. The method consists of a systematic search of the combinatorial "tree" of feasible solutions. Each branch is examined and, with the use of certain tests,

is determined to be either useless for further search or a possible source of an improved solution. The search then continues along the desirable branches.

Several other methods for 0,1 problems have also been introduced including the methods of Glover[10], Lemke and Spielberg[25], and Healy[21].

The foregoing was a brief review of the present status of integer programming. While an attempt was made to present the major techniques of integer programming, several variations and many computer codes for these methods also exist. For a more comprehensive review of the subject, the survey articles of M. L. Balinski[2,3] and E. M. L. Beale[4] are recommended.

So far, the results of computational experience using the existing methods indicate that no single approach is suitable for all integer programming problems. The experience also indicates that problems that are extremely hard to solve by one method may yield readily to another method. Unfortunately there is, at present, no general way to identify, a priori, which problems are most suitable for a given code or, in fact, which problems may be difficult or easy to solve.

In order to investigate further the properties of integer programming it was decided to study a set of problems of identical structure. Since an infinite number of numerical problems can be generated for this structure and since there are so many solution techniques available, it was decided to limit the investigation by only using the all-integer algorithm of Gomory. This selection was made after the method proved acceptable in solving small problems of the given structure. It was also influenced by the availability of

several computer codes of reasonable merit. The problem under study was selected because it has many applications, a few of which are illustrated in the next chapter.

II STATEMENT OF THE PROBLEM

The problem under investigation in this thesis may be stated symbolically as follows:

$$\text{minimize } f(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}x_{ij}$$

subject to the following constraints:

$$(1) \quad \sum_{j=1}^{n} x_{ij} \leq a_i \qquad i = 1,2,\ldots,m$$

$$(2) \quad \sum_{i=1}^{m} p_i x_{ij} \geq b_j \qquad j = 1,2,\ldots,n$$

and (3) x_{ij} is restricted to non-negative integers for all i and j.

Observe that this problem is a special case of the "generalized transportation problem" discussed by Hadley[20] or the "weighted distribution problem" discussed by Dantzig[6]. As pointed out by Hadley, problems of this type do not, in general, yield integer solutions when solved by standard linear programming methods (such as the simplex method).

It should be noted that the problem could be a maximization problem instead of a minimization problem. In addition, the inequality signs of constraints (1) and (2) are somewhat arbitrary; in a given example one or both might be reversed, or one or both might be equality signs.

The following three examples serve to illustrate how problems of

this structure can arise in real situations and to lend physical interpretations to the symbology presented above.

Example 1 (taken from Dantzig[6])

A fleet consisting of m different types of aircraft is to be assigned to n different routes, so as to satisfy passenger demand on all routes at the least operating cost. To formulate this problem in the structure given above, let:

a_i = the number of aircrafts of type i in the fleet

b_j = the number of passengers requiring passage on the j^{th} route

c_{ij} = the operating cost per aircraft of type i when assigned to route j

p_i = the maximum number of passengers carried by a type i aircraft.

x_{ij} = the number of aircraft of type i assigned to the j^{th} route (clearly this must be integer valued).

We may now express the problem as:

$$\text{minimize} \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}x_{ij}$$

subject to

$$(1) \quad \sum_{j=1}^{n} x_{ij} \leq a_i \qquad i = 1,2,\ldots,m$$

$$(2) \quad \sum_{i=1}^{m} p_i x_{ij} \geq b_j \qquad j = 1,2,\ldots,n$$

and (3) x_{ij} must be a non-negative integer.

Thus the objective is to minimize the total operating cost as given in the objective function. The first set of constraints (1) limits the assignment of type i aircraft to the j routes to the maximum number available (a_i). The second set of constraints (2) assure that the capacity of all aircrafts assigned to the j^{th} route will satisfy the demand (b_j) on that route.

Example 2

A cable manufacturer having n different plants is faced with the following problem. Each order for cable must be processed at a single plant location. The manufacturer wishes to assign orders to his plants in such a way that costs are minimized. To set up this problem, let

a_i = 1; that is, order i must be produced once and only once

b_j = the capacity, in feet of cable, of location j for the time period being considered

c_{ij} = the cost of making order i at location j; including both manufacturing and transportation costs

p_i = the size of order i in feet

x_{ij} = 1 if order i is made at location j,
= 0 otherwise.

Thus, assuming m orders for the period under consideration, the problem may be expressed as:

$$\text{minimize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

subject to

$$(1)\quad \sum_{j=1}^{n} x_{ij} = a_i = 1 \qquad i = 1,2,\ldots,m$$

$$(2)\quad \sum_{i=1}^{m} p_i x_{ij} \leq b_j \qquad j = 1,2,\ldots,n$$

and (3) x_{ij} must be a non-negative integer.

In this case, the first set of constraints (1) guarantees that each order is filled once and only once; while the second set (2) assures that the capacity of each plant is not exceeded.

Example 3

A computer system uses a core memory that is divided into various size "holes" by the presence of test words at specific memory locations. The size of a hole is the number of consecutive words available for data storage between test words. The initial data to be stored in this system is available in the form of blocks of various sizes (where size is determined by the number of words in the block). Design requirements specify that data blocks cannot be split and therefore each block must be stored completely in one hole. Because of future growth considerations, it is desired to store the blocks in the smallest holes and save the larger holes for future expansion. Assuming there are m different lengths of data blocks and n holes available, the problem may be stated as follows:

Let,

a_i = the total number of data blocks of length i

b_j = the size of the j^{th} hole. Let the holes be numbered so that $b_1 < b_2 < \ldots < b_n$.

C_j = an artificial cost assigned to each hole. Let the costs be assigned so that $c_1 < c_2 < \ldots < c_n$. Note that this cost is a constant for all i.

p_i = the length of data block i.

x_{ij} = the number of data blocks of length i that are to be assigned to hole j.

Thus the problem may be expressed as:

$$\text{minimize} \sum_{j=1}^{n} c_j x_{ij}$$

subject to

$$(1) \quad \sum_{j=1}^{n} x_{ij} = a_i \qquad i = 1,2,\ldots,m$$

$$(2) \quad \sum_{i=1}^{m} p_i x_{ij} \leq b_j \qquad j = 1,2,\ldots,n$$

(3) x_{ij} must be a non-negative integer.

Here, the objective function is designed to minimize the assignments to the larger holes. The first set of constraints (1) requires that all the data blocks be stored, while the second set (2) assures that the capacity of each hole is not exceeded.

Besides the examples given above, many other problems contain this structure plus additional constraints. Problems in this category include assembly line balancing[32], computer memory allocation[28], new facilities location, and a price break model. The structure was chosen for further study because of this wide applicability. A

search of the literature failed to uncover any specific attempt to solve problems with this structure for integer solutions.

III THE LEXICOGRAPHIC DUAL SIMPLEX METHOD

The purpose of this chapter is threefold. First, the lexicographic dual simplex algorithm will be explained. Second, the methodology of Gomory's cutting plane techniques will be discussed and applied to the algorithm. Finally, the basic methods used in this thesis will be described.

In order to solve linear programming problems, two basic algorithms are generally used. The first of these is the primal simplex algorithm attributed to Dantzig[6]; the second is the dual simplex algorithm developed by Lemke[24]. It has been shown that it is possible for these methods to cycle; that is, to produce the same set of intermediate solutions repeatedly and hence not provide a solution to the problem. In the all-integer programming method the probability of cycling is increased by the fact that only integer numbers are used. For this reason, the lexicographic dual simplex algorithm of Dantzig, Orden, and Wolfe[7] is generally used.

Before explaining the lexicographic dual simplex algorithm, it is helpful to first introduce the concept of the lexicographically positive column vector. A column vector α is said to be lexicographically positive (denoted $\alpha > 0$) if its first non-zero entry is positive. Thus, the vector $\begin{bmatrix} 0 \\ 2 \\ -3 \end{bmatrix}$ is lexicographically positive whereas the vector $\begin{bmatrix} 0 \\ -3 \\ 2 \end{bmatrix}$ is not. Given two column vectors α_1 and α_2 we can say that α_1 is lexicographically greater than α_2 (denoted $\alpha_1 > \alpha_2$) if the vector $(\alpha_1 - \alpha_2)$ is lexicographically positive. For example,

given $\alpha_1 = \begin{bmatrix} 0 \\ 2 \\ 1 \\ -5 \end{bmatrix}$ and $\alpha_2 = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 3 \end{bmatrix}$, we can say that

$\alpha_1 > \alpha_2$ since $\alpha_1 - \alpha_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -8 \end{bmatrix}$

In this thesis, when the terms: less positive, more positive, greater than, larger, smaller, etc., are applied to column vectors, it will always mean in the lexicographical sense.

A convenient way to state an integer programming problem is the following: Find $x_1,\ldots,x_{n+m}$ with all x_i positive and integer valued, in order to maximize

$$x_0 = a_{00} + \sum_{j=1}^{n} a_{0j}(-x_j)$$

when $x_{n+i} = a_{i0} + \sum_{j=1}^{n} a_{ij}(-x_j) \geq 0$ for $i = 1,\ldots,m$.

Observe that the above equations simply express the basic variables x_0, $x_{n+1},\ldots,x_{n+m}$ in terms of the nonbasic variables $x_1,\ldots,x_n$. Actually the variables x_{n+1}, $x_{n+2},\ldots,x_{n+m}$ are the slack variables of the problem and, for convenience of notation, we will hereafter refer to them as x_{s1}, $x_{s2},\ldots,x_{sm}$ respectively.

It is now possible to express the problem in terms of the following tableau:

$-x_1$	$-x_2$		$-x_n$	1		
a_{01}	a_{02}	$\cdots$	a_{0n}	a_{00}	=	x_0
a_{11}	a_{12}	$\cdots$	a_{1n}	a_{10}	=	x_{s1}
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$
a_{m1}	a_{m2}	$\cdots$	a_{mn}	a_{m0}	=	x_{sm}
-1				0	=	x_1
	$\ddots$			$\vdots$		$\vdots$
			-1	0	=	x_n

Basically, a "pivot step" or "iteration" consists of interchanging the roles of a basic and a nonbasic variable. Thus, for example, in the above tableau a pivot step with pivot element $a_{ij} \neq 0$, $i,j \neq 0$, is a step which solves the equation of row i for variable x_j in terms of basic variable x_{si} and the remaining nonbasic variables and uses this equation to eliminate x_j from the remaining equations. Hence, x_j becomes a basic variable as a result of the pivot step whereas x_{si} becomes a nonbasic variable.

The lexicographic dual simplex method requires that all the columns of the tableau with the exception of the constant (or right-hand side) column a_{i0} be lexicographically positive. This requirement makes the problem dual feasible. If the columns are not all lexicographically positive, methods exist for transforming the columns into the proper form (see, for example Balinski[2]). However,

we will always assume that all the columns are positive since, in general, the objective function to be minimized in the class of problems being considered here consists of all positive costs. In addition, for simplicity, we will assume that all the a_{ij} are integers.

The computational process consists of performing a series of systematic pivot steps (iterations) to reach the optimal solution. The optimal solution has been obtained when a pivot step yields a tableau containing all positive and zero elements in the right-hand side column, with the possible exception of a_{00} which represents the value of the objective function. The optimal solution is then the values of the variables as given by the right-hand side column. Each pivot is selected by finding a row i with a negative right-hand side ($a_{i0} < 0$). For this row (i_0) the lexicographic smallest column with $a_{i_0 j} < 0$ ($j \neq 0$) is found. This choice of pivot column maintains the lexicographically positive columns throughout the process. If, in proceeding through the algorithm, a row with $a_{i0} < 0$ is found for which there exists no $a_{ij} < 0$, $j \neq 0$, the problem is infeasible since this represents an inconsistent equation.

Unfortunately, application of this lexicographic dual simplex method to integer programming problems does not, in itself, guarantee an optimal integer solution to the problem. In fact, generally this will not be the case. It is therefore necessary to introduce the cutting-plane technique of Gomory and append it to this method.

The method of Gomory, usually referred to as the all-integer algorithm, differs from the dual simplex method described above in

that instead of continually introducing new nonbasic variables from the original variables of the problem, new ones are created. The new variables are added by first introducing them as basic variables in an additional equation adjoined to the bottom of the tableau. A Gaussian elimination step is next performed to make the new variable nonbasic. After the elimination step, it is possible to remove the adjoined equation from the tableau, since it is not desired to have this new variable re-enter into the solution as a basic variable.

For the sake of completeness, a brief derivation of these new constraints will now be presented. Consider one of the row equations introduced before:

$$(1) \quad x_{si} = a_{i0} + \sum_{j=1}^{n} a_{ij}(-x_j) \geq 0 \text{ or}$$

$$(2) \quad 0 = a_{i0} + \sum_{j=1}^{n} a_{ij}(-x_j) + 1\,(-x_{si}).$$

Every coefficient a_{ij} in this expression, as well as the coefficient of x_{si} can be represented by the form $a_{ij} = b_{ij}\,\lambda + r_{ij}$, where b_{ij} is integer valued, r_{ij} is a remainder term whose value is between 0 and 1, and λ is non-negative. Let us now replace b_{ij} by the symbol $[a_{ij}/\lambda]$ where the square brackets indicate "integer part of". Hence we may write $a_{ij} = [a_{ij}/\lambda]\,\lambda + r_{ij}$; and similarly $1 = [1/\lambda]\,\lambda + r$. Substituting these expressions back into the original equation yields:

$$0 = [a_{i0}/\lambda]\,\lambda + r_{i0} = \sum_{j=1}^{n} ([a_{ij}/\lambda]\,\lambda(-x_j) + r_{ij}\,(-x_j))$$

$$+ ([1/\lambda]\,\lambda + r)(-x_{si})$$

Grouping the remainder terms on the left except for r_{i0} yields:

$$(3)\quad \sum_{j=1}^{n} r_{ij}x_j + rx_{si} = r_{i0} + \lambda\left\{\left[a_{i0}/\lambda\right] + \sum_{j=1}^{n}\left[a_{ij}/\lambda\right](-x_j) + \left[1/\lambda\right](-x_{si})\right\}$$

Note that any non-negative integer values of the variables x_j and x_{si} which satisfy the original row equation will also satisfy this expression. Further, observe that the left-hand expression will always be positive for positive valued variables since all the r_{ij} are positive. Now examine the expression contained in the curly brackets on the right-hand side of the expression, which we will denote as S:

$$(4)\quad S = \left[a_{i0}/\lambda\right] + \sum_{j=1}^{n}\left[a_{ij}/\lambda\right](-x_j) + \left[1/\lambda\right](-x_{si})$$

Note that S will be an integer for integer valued variables, since all the bracketed quantities are, by definition, integers. Further, S is a nonnegative integer because if it were not, the entire right-hand expression given in (3) ($r_{i0} + \lambda S$) would be negative since $r_{i0}<\lambda$. This would make an inconsistent equation since we know the left-hand side of (3) is nonnegative; therefore S must be a nonnegative integer. If we restrict λ to values greater than 1 we may write:

$$(5)\quad S = \left[a_{i0}/\lambda\right] + \sum_{j=1}^{n}\left[a_{ij}/\lambda\right](-x_j)$$

Thus for $\lambda>1$, S will be satisfied by any integer solution to the original linear programming problem. (If $\lambda = 1$ take $S = -x_{si}$).

Gomory presents two requirements for choosing λ; (1) it should produce a pivot element of -1 and (2) it should be as small as possible and still produce the pivot element of -1. The first requirement ensures that the tableau will remain all integer and lexicographically positive after the pivot operation has been performed, while the second requirement attempts to produce the largest change in the objective value.

Thus, when the constraint S is appended to the original problem and pivoted upon, all the computations are carried out through the addition and subtraction of columns, and hence, the tableau always contains only integers. The effect of adding this new constraint to the problem is to introduce a cutting hyperplane into the feasible region. The intermediate solution to the problem then lies on this hyperplane. The procedure of generating hyperplanes and pivoting upon them is continued until the problem is solved or found to be infeasible. Gomory[13] has shown that this procedure will find the optimal solution (if one exists) in a finite number of pivot steps if the rows used to generate the new constraints are chosen properly.

We shall now attempt to summarize the algorithm in more precise terms. In order to do so let us introduce the following notation for convenience in the ensueing discussion. Let J_i be the set of indices j, $j \neq 0$ for which $a_{ij} < 0$ for a particular row i. Therefore J_i is the set of column indices for which row i contains negative elements. In addition let us represent the k^{th} column of the tableau by α_k.

The algorithm begins by selecting a row (i_0) from the eligible rows ($a_{i0} < 0$) of the tableau. Methods for selecting the row will be discussed later, however any row with a negative right-hand side may be selected and therefore these rows are referred to as the eligible rows. Next select the lexicographically smallest α_j with $j \in J_{i_0}$; this will be the pivot column α_{j0}. λ is now determined as follows:

(1) For each j, $j \in J_{i_0}$ find the largest integer μ_j such that $\frac{1}{\mu_j} \alpha_j > \alpha_{j_0}$

(2) let $\lambda_j = \frac{-a_{i_0 j}}{\mu_j}$

(3) find $\lambda = \max_{j \in J_{i_0}} \lambda_j$

Using this value of λ, a new constraint S is generated from row i_0 as described above. The new constraint is adjoined to the bottom row of the tableau and the pivot operation is performed. Upon completion of this process S has been introduced into the problem as a nonbasic variable and the new row is removed from the tableau. This process is then repeated until infeasibility is reached (there exists an $a_{i0} < 0$ for which $a_{ij} \geq 0$ for all $j \neq 0$).

A two variable example problem will now be worked in order to illustrate the methodology. The example problem is not one of the structure being studied in this thesis because a non-trivial two variable problem of that structure cannot be generated. A two variable

problem is presented because it is easy to depict graphically. The example problem is:

$$\text{minimize } f(x) = x_0' = 2x_1 + 3x_2$$

$$\text{subject to } x_1 + 2x_2 \geq 5$$

$$3x_1 + x_2 \geq 6$$

$$\text{and} \qquad x_1, x_2 \geq 0$$

Using asterisks (*) to indicate the row used to generate the new constraint and the pivot column, the solution proceeds as follows:

(0)

$-x_1$	$-x_2$	1		
2	3	0	=	x_0
-1	-2	-5	=	x_3
-3	-1	-6*	=	x_4
-1	0	0	=	x_1
0	-1	0	=	x_2
-1*	-1	-2	=	S_1

(1)

$-S_1$	$-x_2$	1		
2	1	-4	=	x_0
-1	-1	-3*	=	x_3
-3	2	0	=	x_4
-1	1	2	=	x_1
0	-1	0	=	x_2
-1	-1*	-3	=	S_2

Calculations:

$$\mu_1 = \left[\frac{1}{1}\right] = 1 \qquad \mu_2 = \left[\frac{1}{1}\right] = 1$$

$$\mu_2 = \left[\frac{3}{2}\right] = 1 \qquad \mu_1 = \left[\frac{2}{1}\right] = 2$$

$$\lambda_1 = \frac{3}{1} = 3,\ \lambda_2 = \frac{1}{1} = 1 \qquad \lambda_1 = \frac{1}{2},\ \lambda_2 = 1$$

$$\lambda = \max\{\lambda_1, \lambda_2\} = 3 \qquad \lambda = 1$$

$$S_1 = \left[\frac{-3}{3}\right](-x_1) + \left[\frac{-1}{3}\right](-x_2) + \left[\frac{-6}{3}\right] \qquad S_2 = \left[\frac{-1}{1}\right](-S_1) + \left[\frac{-1}{1}\right](-x_2) + \left[\frac{-3}{1}\right]$$

$$S_1 = -1(-x_1) - 1(-x_2) - 2 \qquad S_2 = -1(-S_1) - 1(-x_2) - 3$$

(2)

$-S_1$	$-S_2$	1		
1	1	-7	=	x_0
0	-1	0	=	x_3
-5	2	-6*	=	x_4
-2	1	-1	=	x_1
1	-1	3	=	x_2
-1*	0	-2	=	s_3

(3)

$-S_3$	$-S_2$	1		
1	1	-9	=	x_0
0	-1	0	=	x_3
-5	2	4	=	x_4
-2	1	3	=	x_1
1	-1	1	=	x_2

Calculations:

$\mu_1 = \mu_2 = 1$

$\lambda_1 = 5 = \lambda$

$$S_3 = \left[\frac{-5}{5}\right](-S_1) + \left[\frac{2}{5}\right](-S_2) + \left[\frac{-6}{5}\right]$$

$$S_3 = -1(-S_1) + 0(-S_2) - 2$$

Thus the integer solution to this problem is $x_1 = 3$, $x_2 = 1$ and $x_0' = -x_0 = 9$. Note that the linear programming solution would be $x_1 = 1\ 2/5$, $x_2 = 1\ 4/5$, $x_0 = 8\ 1/5$. The new constraints (S_1, S_2, S_3) may be expressed in terms of x_1 and x_2 as follows:

$$S_1 = -2 + x_1 + x_2$$

$$S_2 = -5 + x_1 + 2x_2$$

$$S_3 = -4 + x_1 + x_2$$

Figure 2 shows graphically how these constraints are added. The solution for each tableau is labeled with the tableau number.

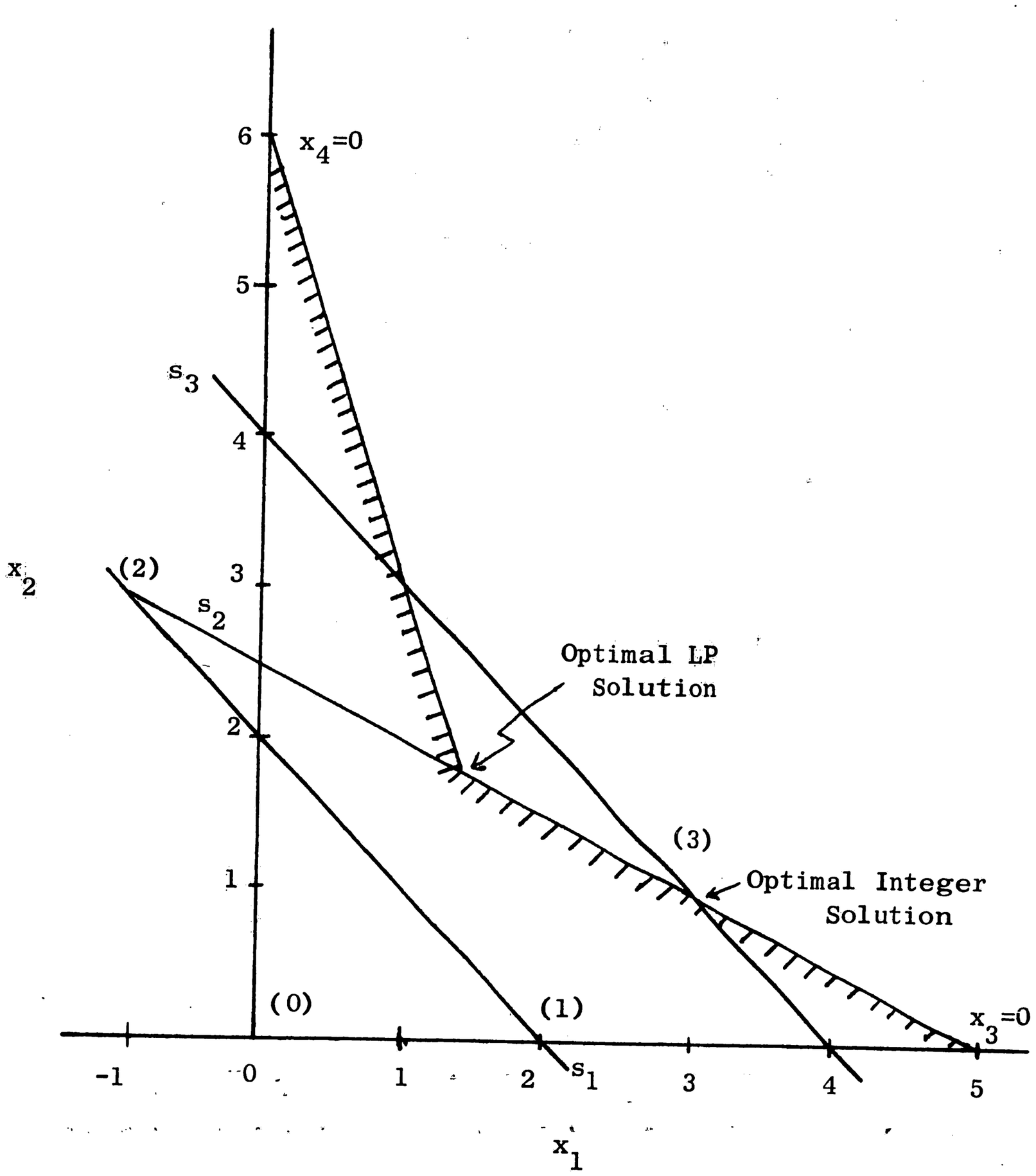


FIGURE 2 GRAPHICAL CUTTING PLANE SOLUTION
TO AN EXAMPLE PROBLEM

As was mentioned earlier, any row with a negative right-hand side may be selected as the row from which the new constraint is generated. The methodology for determining the row to use is generally referred to as pivot selection. Gomory has shown that any pivot selection rule which guarantees that any row with a right-hand side that becomes and remains negative will eventually be selected, produces a finite algorithm. Three pivot selection rules were used to perform the work done in this thesis.

In order to explain these three rules let us introduce one more bit of notation. Let us define I as the set of indices i, $i \neq 0$ for which $a_{i0} < 0$; thus I is the collection of row indices of rows having negative right-hand sides. The following is a description of the three rules.

Rule 1: This rule is the ordinary simplex rule; that is, select the row (i_0) with the largest negative right-hand side. The pivot column is then the least positive eligible column for the selected row. In the event of a tie for the most negative right-hand side, the last such row in the tableau is selected. In summary, the rule is as follows:

(1) Choose i_0 to be the row containing $\min_{i \in I} (a_{i0})$.

(2) In the event that a tie exists after step (1) choose the last such row in the tableau. Thus if we let I_t denote the collection of row indices for which the tie exists, we may write:

$i_0 = \max_{i \in I_t} (i)$.

(3) For i_0 the pivot column (j_0) is chosen as the column that is $\min_{j \in J_{i_0}} (\alpha_j)$.

Rule 2: This rule focuses attention on the columns of the tableau rather than the rows. The rule chooses the eligible row whose pivot column is the largest. Gomory gives two reasons why a pivot selection rule using a column criteria seems desirable. One is that the objective value progresses the most when the most positive column is used. Secondly, the degree of degeneracy is less in the larger column. The rule may be summarized as follows:

(1) Rank the columns of the tableau as 1,2,3,...,n, in order of descending lexicographic size; denoting the rank of the j^{th} column as S(j).

(2) Rank the eligible rows by assigning them the largest rank of their eligible columns, $R(i) = \max_{j \in J} S(j)$.
$i \in I$
Thus the row rank R(i) is the rank of the column that would be the pivot column if row i were selected to generate the new constraint row.

(3) Choose i_0 as $R(i_0) = \min R(i)$.

(4) In the event that two or more rows tie for the minimum rank in step (3), choose the row with the most negative right-hand side. Thus, denoting the tied rows by the set of indices I_t, $i_0 = \min_{i \in I_t} (a_{i0})$.

(5) In the event that two or more rows still remain tied after step (4), choose the last such row in the tableau. Denoting the still tied rows by the set of indices I_s we write: $i_0 = \max_{i \in I_s} (i)$.

Rule 3: This rule, like rule 2, uses a column criteria for the pivot selection. The rule may be stated as follows:

(1) Rank each column of the tableau according to the number of zeroes encountered at the top before the first non-zero element occurs. Thus a rank of zero signifies no zeroes at the top. Denote the rank of the j^{th} column as $Z(j)$.

(2) Rank the eligible rows by assigning them the largest rank of their eligible columns; thus, $R(i) = \max_{j \in J} Z(j)$ $\quad i \in I$

(3) Choose i_0 as $R(i_0) = \min R(i)$

(4) If two or more rows are tied and their rank is zero ($Z(j) = 0$); choose $i_0 = \max_{i \in I_t} (a_{0j})$, otherwise go to step (6).

(5) If a tie still exists after step (4) choose $i_0 = \min_{i \in I_s} (i)$.

(6) If $Z(j) \neq 0$ choose $i_0 = \min_{i \in I_t} (i)$. $\quad j \in J_t$

Appendix A shows a small example problem with the structure given in Chapter II solved by each of the three pivot selection rules.

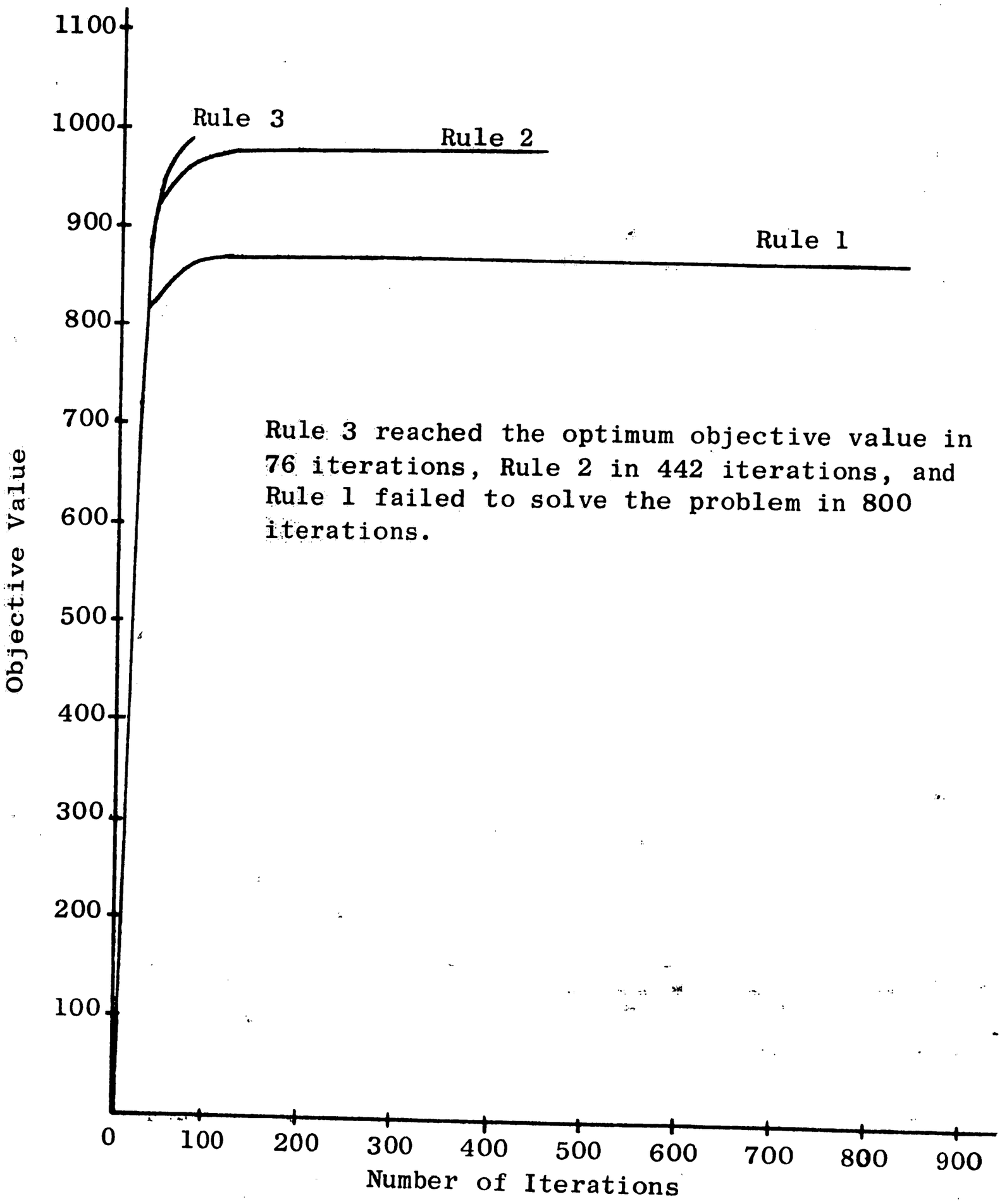
IV "LEARNING" - A HEURISTIC INTEGER PROGRAMMING TECHNIQUE

It was originally intended that the three rules presented in Chapter III be compared experimentally to determine if one rule was universally better than the other two for solving problems of the given structure. After making tests on a group of sample problems, it was determined that this was not the case.

A typical result of this experimentation is illustrated in Figure 3. This figure depicts the value of the objective function for a given test problem as a function of the number of iterations performed by each of the three rules. On another problem, the same type of result would be obtained with the exception that a different rule might solve the problem in the fewest iterations. In fact, by changing only one value in the objective function, problems that were easy to solve by a given rule might become difficult and vice versa. The only general result that could be obtained from this experimentation was that rule 1 did not seem to perform consistently as well as the other two rules. This result agrees with the experimental work reported by Gomory[13] when using this rule.

The next approach taken was an attempt to determine which rule was most likely to solve a given problem before actually attempting to solve the problem. While such an approach may be feasible no way of determining a useful relationship was discovered.

A natural extension of this concept is to determine whether the program can decide which rule it should be using as it works on the problem. Such a procedure is valid in that the rules all use the



Rule 3 reached the optimum objective value in 76 iterations, Rule 2 in 442 iterations, and Rule 1 failed to solve the problem in 800 iterations.

FIGURE 3 PLOT OF OBJECTIVE VALUE VERSUS ITERATIONS FOR A TEST PROBLEM

same tableau and hence strategies can be altered as the solution is proceeding without restarting the process. In fact, methods have been previously employed using various fixed combinations of pivot selection rules in attempts to solve integer programming problems[28]. Since such a method seemed plausible, it was necessary to determine a programmable criteria on which this rule selection could be determined. Figure 3 provides a clue to at least one possible criteria: The objective function. Note that as the number of iterations increases the rat of change of the objective function decreases. This is to be expected since, in general, the columns of the tableau become less positive as the method progresses towards a solution. It appears that the rule we are seeking is the one in which this rate of change decreases at the slowest rate.

Thus it would appear that a good approach would be to begin the problem by using all three rules concurrently for some number of iterations, that is, solve the problem simultaneously using each of the three rules until it becomes clear which rule was progressing the best. Then the other two methods would be abandoned and the solution continued using the remaining rule. Unfortunately, such an approach has several serious drawbacks. First, it would increase the size of the memory required to perform the computations. Since this would reduce the size of the problem that could be processed by the program, this is a definite disadvantage. In addition, the time required to process a problem would be increased since all the initial iterations would require three times the computations. Finally, the question of determining how many iterations are required to select the

best rule is extremely difficult to answer. This problem is compounded by the fact that rule 1, which solved the least test problems of the three rules, generally produces the fastest change in the objective function in the early iterations. This is not surprising since rule 1 always selects the most negative right-hand side. Figure 4 shows the progress of the problem illustrated in Figure 3 for the first 100 iterations in more detail.

The figure serves to highlight the difficulties of the rule selection problem. Observe that rule 1 exhibits the best rate of change for at least the first five iterations and produces the highest objective value for the first 24 iterations. Note also that rules 2 and 3 produce identical results for the first 10 iterations and are virtually inseparable until rule 3 solves the problem on the 76 iteration. Observe, finally, that the "best" rule as determined by the best rate of change varies dependent upon what iteration of group of iterations is used to compute the rate of change. Thus, for example, between iterations 15 and 20 rule 1 has the best rate of change, but between iterations 18 and 20 rule 2 has the best rate of change.

Because of the difficulties mentioned above, it was decided that an alternative approach would be taken. The approach is to allow each rule to work on the same tableau in turn, thus elminiating the increased memory and time problems discussed above. This approach, however, does create several additional problems. Perhaps the most difficult of these is how to determine the best rule at any given time.

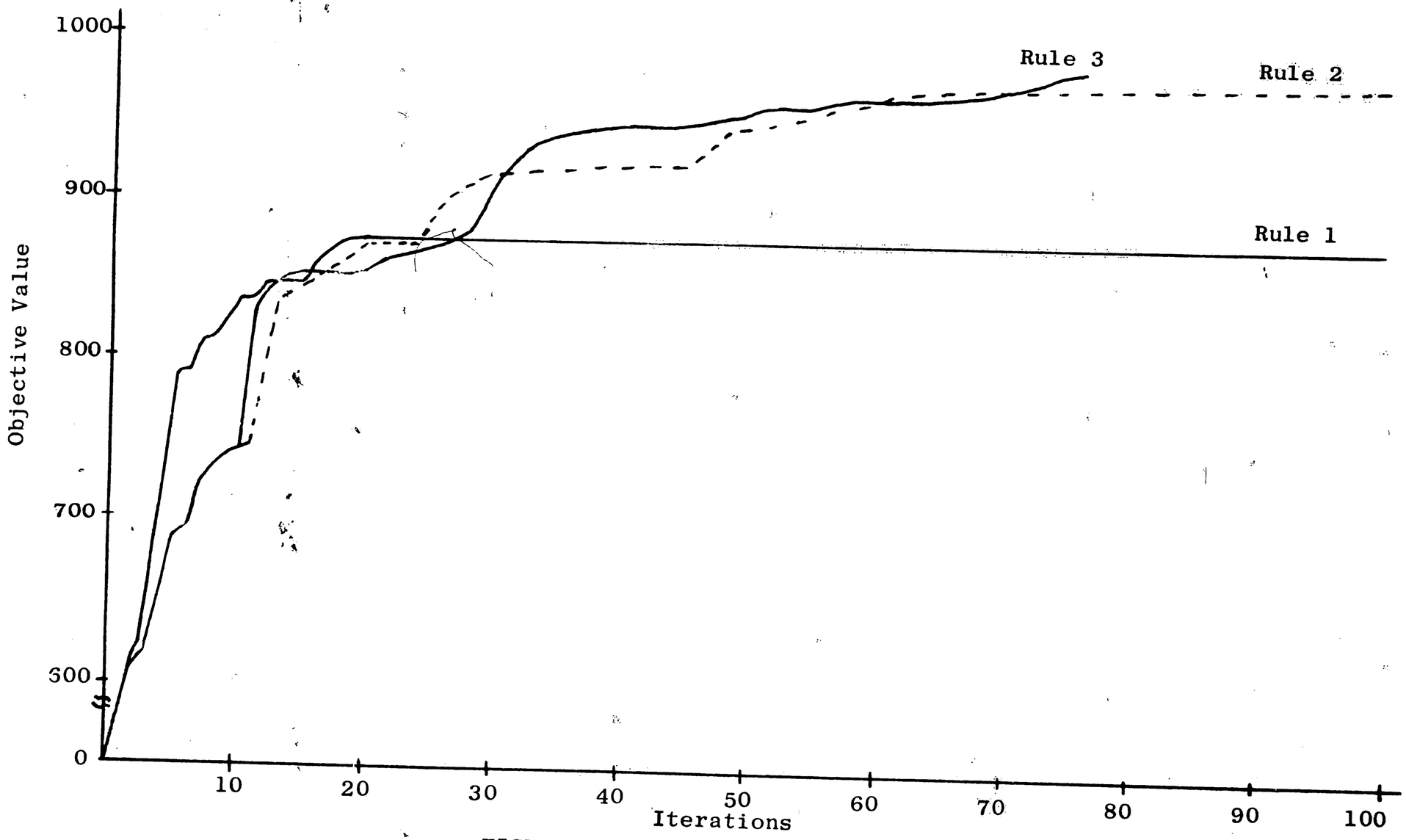


FIGURE 4 EXPANDED PLOT OF FIGURE 3

This difficulty occurs because the rate of change, in general, decreases as the number of iterations increase. Hence, this method is biased in favor of the rule which is used first, since that rule is most likely to cause the greatest improvement in the objective function. The problem of how many iterations to allow before determining a best rule is now compounded by the fact that the more iterations a rule is allowed to perform the more likely it is to appear better than the rule(s) that succeed it.

Since the rate of change of the value of the objective function is a function of the number of iterations, it is extremely difficult to determine a criteria by which to select the "best" rule. If a method existed for determining the expected value of the objective function for each iteration, then a comparison with the value actually obtained might uncover the desired rule. Lacking such a method, the following technique was used.

Each rule was allowed to run in turn until it no longer improved the objective value for a prescribed number of iterations (that is, the rate of change of the objective value remained zero for a fixed number of iterations). The rule's would be run once each in a prescribed sequence and then the sequence would be reversed and the process repeated. The purpose of reversing the sequence is to reduce the afore-mentioned biasing effect of the sequential processing. Upon completion of the two cycles, the rule that had been used for the most iterations would be selected as the "best" rule. We shall refer to this rule selection process as "voting".

Having established this criteria for rule selection, several parameters remain to be determined. First it is necessary to determine if such a method is dependent on the initial sequence of rules and, if so, what sequence should be used. Second, it must be decided for how many iterations the objective value should remain constant before the next rule in the sequence is used. We will call this number of iterations the "learning value". In order to determine some experimental answers to these questions, some of the problems originally used to test the three rules were used. A series of different combinations of rules and learning values were tried in an effort to find a reasonable set of parameters.

One of the interesting observations obtained from this investigation was the general ineffectiveness of rule 1. While this rule did increase the objective value the most at the very beginning of a problem it seemed to be of little value thereafter. Based on this observation it was decided to investigate the effect of (1) only using rule 1 at the beginning of the problem and (2) not using rule 1 at all.

The original investigation also showed that the method was rule sequence dependent, but not in a predictable fashion. On a given problem a particular sequence would be best, but on another problem another sequence might be best. Thus, this result was quite similar to the initial results obtained using the three rules. In a like manner the method also appeared to be "learning value" dependent, with no one value exhibiting consistent superiority.

When the effect of using rule 1 only at the beginning of the problem was studied it was found that the rule seemed to generally help if it were only used a few times and the sequence that followed it was rule 2 followed by rule 3; but it produced more erratic results when the succeeding sequence was rule 3 followed by rule 2. If rule 1 was used for a large number of iterations it generally produced bad results. When the rule was not used at all, the best sequence seemed to be rule 3 followed by rule 2. However, the complete omission of the rule did not, in every case, produce better results than when it had been included.

A final observation that could be made from this set of experiments was that the rule selected after the initial voting procedure was not always the rule that had performed the best on that problem. In general this seemed to be because the bias caused by the sequencing had not been overcome. It was therefore decided to allow for further voting to be done as the method progressed.

In all, the above mentioned experimentation consisted of using 41 test problems. Most of the learning experimentation was done using 13 of these problems which were processed a total of approximately 400 times. Based on this experimentation a method of learning was selected as the most promising for further investigation. It was decided that within this method two rule sequences also merited additional consideration.

The learning method employed uses only rules 2 and 3 in the basic learning and voting procedure with r le 1 used only in special situations. The two different r le sequences are shown in Figure 5

and 6. Figure 5 depicts the method using the initial rule sequence: rule 3 followed by rule 2; we will denote this method by L_{32}. Figure 6 depicts the sequence: rule 2 followed by rule 3, in chich case rule 1 proceeds the initial sequence for four iterations; we will denote this method by L_{123}. With the exception of this use of rule 1 in L_{123} and the fact that the rule sequences are initially reversed in the two techniques, the two basic methods are the same. In both cases, the methods proceed as follows.

Two cycles of the preselected sequence are run, with the sequence reversed for the second cycle. The learning value for these initial cycles is "1". This value was selected bacause it requires the least possible number of iterations to complete the cycles. The voting procedure is then used to determine the new sequence and learning values to be used as shown in the figures. The first rule to be used in the new sequence is the rule that was used most frequently in the first voting sequence. In the event of a tie a new two cycle sequence is established. The learning value is varied as a function of the success of the rules. In cases where both rules advance the value of the objective function during the previous voting sequence, an equal learning value is used for both rules in the new sequence. In cases where only one rule advances the value of the objective function during the previous voting sequence that rule is assigned a larger learning value than the unbeneficial rule. This is done because it is assumed that the beneficial rule is the best rule to use on the problem. When neither rule is found to improve the objective function, rule 1 is used with a small learning value, followed by a

R1,R2,R3 = Rule 1, Rule 2, Rule 3
N = Number of Cycles Performed
L = Learning Value
I( ) = Number of Iterations of Indicated Rule Performed Since Last Comparison

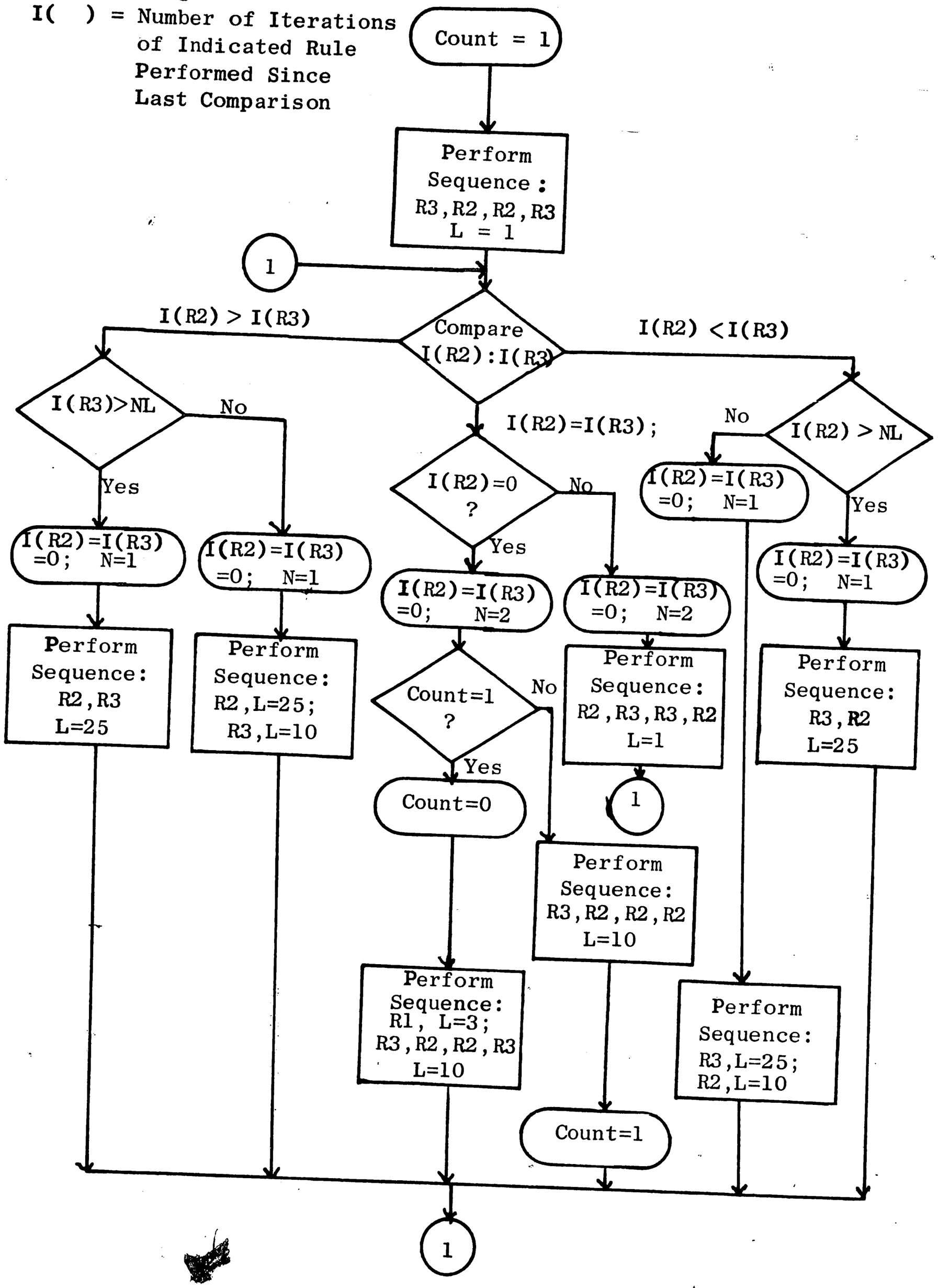


FIGURE 5 FLOW CHART OF L_{23}

R1,R2,R3 = Rule 1, Rule 2, Rule 3
N = Number of Cycles Performed
L = Learning Value
I( ) = Number of Iterations of Indicated Rule Performed Since Last Comparison

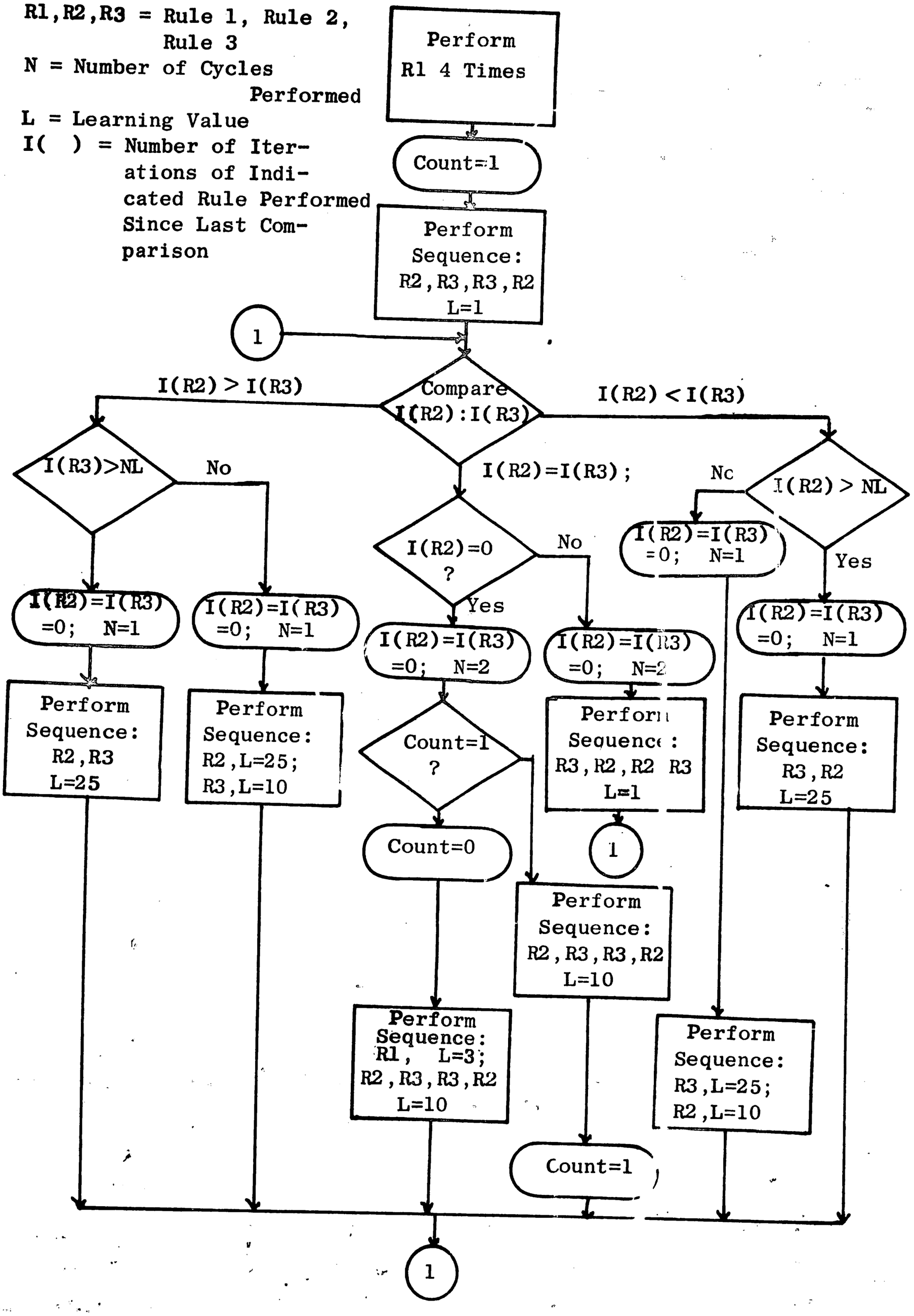


FIGURE 6 FLOW CHART OF LEARNING METHOD L_{123}

two cycle sequence of the two rules at a different learning value. Since rule 1 seldom improved the objective value it is only used every other time this condition occurs; however, its use in this fashion has proved helpful in causing the other rules to become effective again in some problems.

Chapter V describes an experiment used to compare the two learning methods with the three basic rules.

V DESIGN OF AN EXPERIMENT TO TEST THE VARIOUS TECHNIQUES

In order to conduct an experiment to test which of the aforementioned integer programming methods is best, it was necessary to define the following four factors: (1) a set of test problems, (2) a computer program to solve these problems by each of the methods, (3) a criterion for determining which method is the most effective, and (4) a method for evaluating this effectiveness.

In order to generate the test problems necessary to perform this experiment, the following assumptions were made:

(1) The problems would be of the size m = 3, n = 4 (that is, 7 constraints with 12 unknowns). This size was selected because it provides non-trivial problems while keeping the effort to prepare the problems at a minimum.

(2) All the given constants of the problem (a_i, b_j, p_i, and c_{ij}) are positive integers.

In order to establish a representative set of problems consistent with the assumptions given above, a random number table[5] was used to select the values of the constants. Each problem was generated by first assigning an arbitrary range of values for three of the constants (a_i, p_i, and c_{ij}). Values for these constants within the specified ranges were then selected from the random number table. After this selection was made, a range of values was assigned for the remaining constant (b_j) and values selected for it from the random number table. The range of values for this constant was selected in

such a manner that a feasible problem with reasonably tight constraints was likely to result. The generation of problems in this manner was necessary to produce a set of realistic problems. When random numbers were employed without establishing any ranges for the constants, infeasible or trivial problems almost always resulted.

In all, a set of 25 feasible test problems were generated (infeasible problems were rejected). The problems, as well as the ranges specified in their generation, are given in Appendix B.

The program used to solve these test problems is a Fortran IV program known as IPWE (Integer Program-Western Electric). The program is based upon IPSC which is the work of R. E. D. Woolsey and C. A. Trauth[34].

The basic differences from IPSC are that IPWE includes the learning technique and that it stores only the nonzero elements of the tableau, thus allowing for larger size problems to be attempted. The problem size that IPWE can operate on is determined by the following relationship:

$$d(r + c)(c) \leq 100,000$$

where :

r is the number of rows,

c is the number of columns, and

d is the ratio of nonzero elements to all elements.

IPWE contains the three pivot selection rules as subroutines and allows for the selection of any single rule, any combination of the rules, or the learning method through the use of a single control

card. The programming of these features as well as some modifications to increase the basic program's speed are the work of J. H. Schmaltz and the author.

The output of the program includes the optimal value of the objective function, the values of the decision variables, the number of iterations required to solve the problem, the time required to solve the problem, and the number of iterations each pivot selection rule was used during the solution of the problem.

All the computations for this experiment were performed using IPWE on an IBM 360/50 computer.

Before the test problems could be processed and the result evaluated, it was necessary to determine a criterion on which to make this evaluation. At least three measures exist which can be used to evaluate the relative effectiveness of the methods:

(1) the number of problems solved by the method,

(2) the number of iterations required to solve the problems, and

(3) the time required to solve the problems.

Actually, all three of these criteria are related to each other. Clearly, the time required to solve a problem is a function of the number of iterations required. In addition, if an upper limit is set as to how many iterations or how much time will be allowed for the solution of any one problem, then the number of problems solved will also be a function of these measures. An upper limit of this sort is necessary in order to keep the amount of computer time expended at a tolerable level.

In general, what is of interest is the method that solves the most problems in the least time, however, it is possible that none of the methods exhibit both of these properties. For example, one method may always be the fastest on the problems it solves, yet fail to solve many problems. Therefore, including any unsolved problems in with the solved problems should provide a good measure of each method's overall effectiveness.

While the time required to process the problems might seem the best measure to use, the time is a function of the computer used and of how efficiently each method is programmed. The number of iterations required to solve a problem is a constant regardless of these factors. Hence, it was decided that the methods would be statistically evaluated using both criteria (2) and (3) above. With 25 test problems, it was unlikely that criterion (1) could produce statistically significant results. However, as mentioned above, these results are included in the measures of (2) and (3).

In order to conserve computer time, a pivot limit of 1,000 iterations was established for each problem. Hence, an "unsolved" problem is one that was not solved in 1,000 iterations. While a higher pivot limit could have been selected, it was felt that a larger value might bias the results so that the problems that solved readily would have no effect on the final outcome, since they would have been completed in many fewer iterations than the problems that did not solve. The 1,000 iteration limit was also selected because past experience indicated that most problems of this type were solved within the first

700 iterations or else ran for a much larger number of iterations without solving.

Having established a set of test problems and two criteria for judging which method(s) performs best on the problems, it is possible to discuss statistical methods for evaluating the results of the experiment. The results will be the number of iterations required and the time required to solve each problem by each method. Identical statistical analysis is to be made on both the iterations and time required, hence only the iterations will be considered in this discussion. After processing the problems, the mean number of iterations required to process the problems by each method can be determined. Thus, it is possible to test the hypothesis that these five means are equal. Such a test can be performed by analysis of variance techniques. Analysis of variance consists of partitioning the total sum of squares of deviations from the mean into component sums of squares each associated with a particular factor or with experimental error, and similarly partitioning the total number of degrees of freedom. Statistical tests can then be performed to determine which factors, if any, are significant when compared to the experimental error. In this experiment there are three factors to be considered: the methods, the problems, and the interaction between the methods and the problems. The methods are the factor of interest in this experiment since the hypothesis to be tested is that there is no difference in the means produced by the five methods. The problem effect is likely to be significant since there is liable to be a large

variation between problems, with some problems being easy to solve and others being unsolvable. The interaction between the problems and the methods is assumed small and, lacking any other source of experimental error, must be considered the error term. Since it was not possible to determine a priori which method would perform well on any given problem and the problems were selected at random, it will be assumed that this interaction is a random variable. In order to use analysis of variance it is further necessary to assume that the error is normally distributed. No evidence can be given to support or refute this assumption; however, it is likely that using the pivot limit of 1,000 iterations tends to disrupt normality. However, the fact that means are being compared tends to increase the probability of a normal distribution, as shown by the central limit theorem.

Based on the above discussion, the following two statistical tests are proposed:

(1) A two-way classification analysis of variance to test the hypothesis that the means are equal, and

(2) Duncan's New Multiple Range Test[8].

Since certain of the assumptions necessary for the rigorous application of these tests have not been met, the tests provide only an approximate procedure; however, the approximation seems the best available. Thus, these tests are offered only as an attempt to lend additional insight into the results obtained.

Analysis of variance uses the F test to test the null hypothesis.

If the hypothesis is rejected, however, such a test gives no decision as to which of the means are significantly different. Duncan's test will show which of the means are significantly different as well as any natural subgrouping of the means which may exist. The test is a compromise between performing only one t-test on all the means and performing t-tests on all the possible combinations of the means.

For a given significance level α, the actual test is made at the significance level α_p, where $\alpha_p = 1 - (1-\alpha)^{p-1}$ and p is the number of means which lie between the two means under consideration, including those two means.

VI RESULTS

General

The results of attempting to solve the 25 test problems by each of the five methods are contained in Tables 1 and 2 below. Table 1 shows the number of iterations required to solve each problem (with the pivot limit of 1000 iterations), while Table 2 shows the results in terms of the times required to process that number of iterations by each method. The tables also show the total and mean number of iterations and times, respectively. In terms of the number of iterations required, method L_{32} required the fewest iterations, followed in order by rule 2, rule 3, rule 1, and L_{123}. The order for the time required was the same with the exception that rule 1 ranked above rule 3.

There were four problems (numbers 7, 21, 24 and 25) that no method solved. In terms of the total number of problems solved, L_{32} again appears best, having solved 20 of the problems. Rule 2 solved 19 problems, followed by L_{123} which solved 18 and rules 1 and 3 which each solved 16. Table 3 below summarizes these results in terms of the percent of problems solved by each method. The table shows these results in terms of all 25 problems and in terms of the 21 solved problems.

TABLE 1

Results in Terms of Iterations Required

Problem Number	Methods				
	Rule 1	Rule 2	Rule 3	L_{32}	L_{123}
1	10	10	11	11	10
2	13	13	13	13	13
3	4	4	4	4	4
4	25	17	14	14	28
5	15	16	19	17	18
6	17	914	639	51	13
7	1000	1000	1000	1000	1000
8	104	122	1000	1000	124
9	17	15	21	19	22
10	130	83	124	124	83
11	13	13	14	14	13
12	1000	716	29	29	693
13	1000	1000	1000	576	432
14	620	1000	24	24	947
15	6	6	6	6	6
16	27	798	39	26	993
17	80	27	33	30	27
18	1000	55	1000	134	1000
19	1000	64	1000	65	1000
20	31	18	20	21	17
21	1000	1000	1000	1000	1000
22	1000	879	1000	64	1000
23	52	23	26	26	24
24	1000	1000	1000	1000	1000
25	1000	1000	1000	1000	1000
TOTAL	10164	9793	10036	6268	10467
AVERAGE	406.56	391.72	401.44	250.72	418.68

TABLE 2

Results in Terms of Time (in Seconds) Required

Problem Number	Methods				
	Rule 1	Rule 2	Rule 3	L_{32}	L_{123}
1	2.75	2.86	2.93	3.01	2.80
2	2.25	2.67	2.73	2.75	2.56
3	0.92	1.00	1.00	1.03	0.93
4	2.89	2.57	2.36	2.38	2.58
5	3.11	3.09	2.63	3.18	2.54
6	2.06	152.82	75.19	8.23	1.73
7	111.59	91.34	114.18	90.83	88.59
8	14.40	19.30	155.36	149.29	22.96
9	2.63	2.45	2.91	2.74	2.73
10	16.90	10.22	16.59	16.53	9.78
11	2.58	2.57	2.59	2.75	2.47
12	170.94	117.51	4.43	4.49	121.01
13	117.42	105.68	86.91	72.20	48.42
14	46.44	81.94	2.93	3.03	84.12
15	1.31	1.27	1.30	1.37	1.33
16	4.24	116.11	7.32	5.11	149.55
17	6.82	3.13	3.60	3.44	3.13
18	149.53	7.01	141.84	20.75	154.63
19	116.50	11.67	168.65	11.21	166.89
20	3.60	2.75	3.02	3.24	2.53
21	108.84	92.04	141.67	116.83	142.47
22	159.68	114.63	112.22	10.47	131.34
23	5.21	4.00	4.64	4.56	4.09
24	88.07	77.16	106.96	95.61	90.16
25	61.20	68.71	102.01	98.68	76.65
TOTAL	1201.88	1094.51	1266.02	733.70	1315.98
AVERAGE	48.07	43.78	50.64	29.34	52.63

TABLE 3

Percent of Problems Solved

Method	All 25 Problems	21 Solved Problems
L_{32}	80	95.4
Rule 2	76	90.5
L_{123}	72	85.8
Rule 1	64	76.3
Rule 3	64	76.3

A further study of Table 1 shows that the problems can be subdivided into several categories of interest. First, as mentioned previously, problems 7, 21, 24 and 25 form the set of unsolved problems. While this set is of little interest for determining which method was the best, it may be very useful in determining how these methods can be improved; however that subject will be deferred until Chapter VIII. Another set of problems of interest is the group of fourteen problems that all five methods solved. This set consists of problems 1-6, 9-11, 15-17, 20 and 23. The remaining seven problems: 8, 12-14, 18, 19 and 22, are the problems that were solved by at least one, but not all, of the methods. Finally, by combining these last two sets, there is the set of twenty-one solved problems. Tables 4 and 5 summarize the average number of iterations and times required to solve each of these four sets of problems as well as all the problems.

TABLE 4

Average Number of Iterations for Various Sets of Problems

	Rule 1	Rule 2	Rule 3	L_{32}	L_{123}
All Problems	406.56	391.72	401.44	250.72	418.68
21 Solved Problems	293.52	275.85	287.42	108.00	307.95
14 Problems Solved by all Methods	31.42	139.78	70.21	26.85	90.78
7 Problems Solved by at leastone Method	817.71	548.00	721.85	270.28	742.28
4 Unsolved Problems	1000	1000	1000	1000	1000

TABLE 5

Average Processing Times for Various Sets of Problems

	Rule 1	Rule 2	Rule 3	L_{32}	L_{123}
All Problems	48.07	43.78	50.64	29.34	52.63
21 Solved Problems	39.62	36.44	38.15	15.79	43.71
14 Problems Solved by all Methods	4.09	21.96	9.20	4.30	13.48
7 Problems Solved by at least one Method	110.70	65.39	96.04	38.77	104.19
4 Unsolved Problems	92.42	82.31	116.20	100.48	99.46

A study of Table 4 lends additional insight into the results obtained. Excluding the group of four unsolved problems, method L_{32} required the fewest iterations in each group. On the 21 solved problems, the second best method (rule 2) averaged over 2 1/2 times the number of iterations averaged by L_{32}. Similarly when the 7 problems solved by at least one method are examined, L_{32} used fewer than

1/2 the iterations that the second best method (again rule 2) used. On the 14 problems solved by all methods, however, L_{32} was only slightly better than rule 1. This brings out an interesting point; rule 1 was second best on the 14 easy problems, but worst on the 7 harder problems. Similarly, rule 2 was second best on the harder problems, yet worst on the 14 easy problems. Little difference is seen between rule 3 and L_{123}, although rule 3 is slightly better in each group. Table 5 shows a similar pattern for the five methods in terms of the average time required. The one main difference is that rule 1 averaged .21 seconds less than L_{32} on the fourteen easy problems, even though it had averaged over 4 iterations more than L_{32} on those problems. Thus, from these results, it appears that rule 1 is very good on the easy problems whereas rule 2 is fairly good on the harder problems. However, L_{32} is very good in both these areas. In fact, L_{32} solved 19 of the problems in less than 150 iterations each. None of the other methods solved more than 15 problems in 150 iterations and only rule 2 solved 19 problems in the entire experiment. Similarly it can be said that L_{32} solved 19 of the problems in less than 21 seconds and no other method solved more than 15 in a like amount of time.

Table 6, below, shows the average iterations per second that each method performed on each of the above mentioned groups of problems. The table shows that the methods are fairly competetive on this basis. The main reason that the averages are higher on the four unsolved problems is believed to be that the times calculated include the total

processing time for each problem, thereby including time that is taken at the beginning and end of the problem for bookkeeping purposes (as well as the time necessary to calculate the processing time). On the longer running problems this time is spread over many more iterations than on the easier problems. In addition it is believed that this phenomenon may in part be a measure of the degeneracy of the problem. This is because the time required by IPWE to store only the non-zero elements after each iteration would be less on degenerate problems since fewer elements would change from zero to non-zero and vice versa.

TABLE 6

Iterations/Second on the Various Sets of Problems

	Rule 1	Rule 2	Rule 3	L_{32}	L_{123}
All Problems	8.45	8.94	7.92	8.54	7.95
21 Solved Problems	7.40	7.56	7.53	6.83	7.04
14 Problems Solved by all Methods	7.68	6.36	7.62	6.23	6.73
7 Problems Solved by at least one Method	7.38	8.38	7.51	6.97	7.12
4 Unsolved Problems	10.81	12.14	8.60	9.95	10.05

Statistical Tests

The analysis of variance (ANOVA) tests and Duncans New Multiple Range Tests were performed on the data given in Tables 1 and 2. The results of the ANOVA tests are shown in Tables 7 and 8 respectively.

TABLE 7

Anova Test on Iterations (Table 1)

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Squares	F Ratio
Methods	483030.1	4	120757.5	1.71
Problems	17860244.0	24	744176.7	10.55
Interaction	6790395.0	96	70733.3	-

TABLE 8

Anova Test on Times (Table 2)

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Squares	F Ratio
Methods	8651.3	4	2162.8	1.42
Problems	227386.8	24	9474.5	6.2
Interactions	146561.2	96	1526.7	-

Since an F ratio of 2.01 or more is necessary to show significance at the .1 level, the method effect cannot be shown to be significant. (The problem effect is significant at the .001 level for both tables). The ANOVA tests were also run on the 21 test problems that were solved by at least one rule, thus excluding the 4 unsolved problems. No differences in the F ratios occurred in these tests. The reason that the F level for the method effect is not significant is most likely the fact that the interaction (error) term is so high. This term is high because there exist large differences between how the methods solved certain problems (for example problems 6, 8, 12, 13, 14, 16, 18, 19, and 22) and because none of the methods was universally better than the others on all problems.

Another reason why no difference in methods was determined is that analysis of variance is not a good technique for separating out one method from a group of methods that have similar means as in the results given here. In light of this fact and since L_{32} appeared better than the other rules, it was decided to compare it individually with each of the other methods. It is important to realize, however, that such a comparison increases the probability of declaring two means significantly different when they are, in fact, equal. The results of these tests, when run on the number of iterations required, showed L_{32} significantly better than each of the other methods at the .1 significance level, and better than rule 3 at the .05 level. When similar tests were performed on the times required, method L_{32} was better than rule 3 at the .05 level and better than L_{123} at the .1 level. However, no significant difference could be shown between L_{32} and rules 1 and 2.

When the data of Table 1 was analyzed using Duncan's test, method L_{32} was better than the other four methods at the .1 significance level, and better than L_{123} at the .05 level. When Duncan's test was used on the times required (Table 2), L_{32} was superior to rule 3 and L_{123} at the .1 level, but not to the other 2 methods.

While these statistical tests do not show the strong significance that is usually desired (at the .05 or .01 significance levels) in such tests, they do indicate that method L_{32} is the best of the 5 methods, particularly in terms of the number of iterations required. In addition, if the pivot limit were increased, L_{32} is likely to compare even more favorably with the other methods both in terms of

iterations and time, since it has solved the most problems.

iterations and time, since it has solved the most problems.

VII CONCLUSIONS

The work performed in this paper has involved attempting to find a suitable method for solving the generalized transportation problem for integer solutions. Such a method would be valuable because problems of this nature are frequently encountered in industrial applications. In particular, this investigation focused on the all-integer integer programming algorithm of Gomory. Three commonly used pivot selection rules and two variations of the "learning" method advanced by the author were compared on twenty-five randomly selected test problems to see which, if any, was best at solving such problems. The results of this comparison showed the learning method denoted by L_{32} was the best of the five methods. This judgement is based on the fact that this method solved the most problems, required the fewest number of iterations and the least processing time. In addition, statistical tests also indicated that this method was better than the others.

It should be noted that in the solution of the test problems, some of the problems were solved by the learning techniques before the first voting procedure could be accomplished. Of course, this simply shows that the learning technique is forced to be competitive with the basic rules on the simple problems. Thus, in those cases, the learning technique produces the same results as the first rule used in the voting sequence. In addition, it frequently occurred that the rule selected after the first voting was performed was not the rule which had performed the best individually on the problem. In general, this seemed to occur when the biasing effect discussed in Chapter IV was

not overcome. If methods can be devised to overcome this effect still greater benefits may occur. However, even when this "incorrect" rule selection occurred, the learning method still frequently caused an improvement in the number of iterations required.

It is important to realize that while the learning technique developed here was applied only to one class of problem, there is nothing in its development that restricts its application to only that class of problem. In fact, there is no reason why this technique cannot be applied to any integer programming problem, regardless of its structure. Three large problems of different structures have been processed using this learning method. The method solved one problem of 25 variables and 48 constraints in 23 seconds and another of the same size in approximately 70 minutes. The method failed to solve a problem of 189 variables and 47 constraints in 60 minutes; however, it did advance the objective value of this problem further than any of the three pivot selection rules in the same amount of time. More computational experience is necessary before any statement can be made on the learning techniques effectiveness on problems other than the structure used in this paper.

Overall, the work performed here may be of more value to the field of integer programming in general, than to the solution of the specific problem considered. While the learning method failed to solve every test problem within the 1000 iteration limit, it did solve 80% of the problems, which is, even for small problems, a good average in the integer programming field. The use of the learning technique has

shown that it is possible to use combinations of pivot selection rules to great advantage. Some possible ways to improve on this technique are discussed in the next chapter.

VIII RECOMMENDATIONS FOR FURTHER STUDY

Several aspects of the learning technique are open for further investigation. First, it may be possible to improve upon the learning values used here. The four test problems that were not solved by any of the methods may be of particular interest in such an effort. Perhaps other learning values may prove better for problems of a different size or structure. In fact, it may be that these parameters should be changed according to specific characteristics of the problem to be solved, such as the problem size or the percentage of zero elements in the problem.

Only three pivot selection rules were used in the work performed. Thus many other rules remain to be studied. Additional study may yield a better set of rules to be used in the learning process.

Recently, much work has been done to improve the method of generating the new constraints required by the all-integer method. Glover[11], Gomory[16], and Wilson[33] have all recently discussed techniques for generating stronger cutting planes than those originally proposed by Gomory. Introduction of these techniques into the learning program should generally reduce the number of iterations required to solve problems. Whether or not these techniques would also reduce the time required to solve problems requires computational experience.

Another area open for investigation is that of determining a different and perhaps more sophisticated learning technique. It may be possible to make better use of the rate of change of the objective

function; or perhaps a learning technique that does not rely on the value of the objective function can be developed. Other methods might involve the use of all the right-hand side elements, such as learning on the size of the largest negative element or on the number of negative elements present. Another possibility would be to learn on the time required to process each iteration. Due to the nature of IPWE, pivot columns that contain a large number of zeroes should be processed faster than less degenerate columns. Thus it may be possible to establish a criterion for learning by monitoring the time required for each iteration and switching rules whenever the rule in use performs an iteration in less than some pre-established base time for that rule. Unfortunately this base time will be dependent on the problem size. In addition, such a method requires the computing of the time after each iteration, which may slow up the method too much to prove useful. Of course, the more involved the method, the more likely it is to be more time consuming than the simple learning method used in this thesis. Thus it may be necessary to make a trade off between complexity and computer time.

More computational experience is required in order to evaluate the learning technique's applicability to larger problems and to problems not of the structure presented here.

A careful and thorough study of the test problems (Appendix B) used in this thesis may give insight into how to determine, a priori, which method to use on a given problem. Finally, further study of the problem structure presented here may uncover still better methods of solving it for integer solutions.

APPENDIX A

An Illustrative Example Solved by the Three Pivot Selection Rules

In order to further illustrate the methodology of the lexicographic dual simplex technique and the three pivot selection rules described in Chapter III, the following example of the problem structure given in Chapter II will be solved by each of the three rules:

$$\text{Minimize} \quad X_0 = X_{11} + X_{12} + 3X_{13} + 6X_{21} + 4X_{22} + 7X_{23}$$

subject to:

$$X_{11} + X_{12} + X_{13} \leq 7$$

$$X_{21} + X_{22} + X_{23} \leq 5$$

$$3X_{11} + 4X_{12} \geq 16$$

$$3X_{12} + 4X_{22} \geq 6$$

$$3X_{13} + 4X_{23} \geq 10$$

and X_{ij} must be non-negative and integer valued.

For each of the three rules, a brief summary of the rule is presented followed by the iteration by iteration solution to the problem using that rule.

For all three rules the initial tableau (labeled iteration 0) is identical since it contains the initial statement of the problem. In each tableau, the row used to produce the new constraint and the pivot column will be denoted by asterisks. The tableau also shows the new constraint generated from the selected row with the value of lambda (λ) given below it. The result of the pivot operation on the new constraint is then shown in the next tableau and the process is repeated until the optimum solution is reached.

Rule 1 - Rule one selects the row with the most negative right-hand side as the row from which to generate the new constraint. In the event that two or more rows tie for most negative right-hand side, the lowest such row in the tableau is selected. (Iterations 6 and 7 of the example problem illustrate this). As always, the pivot column is the lexicographically smallest of the eligible columns of the selected row. The solution to the example problem using rule 1 proceeds as follows.

RULE 1

(0)

$-x_{11}$	$-x_{12}$	$-x_{13}$	$-x_{21}$	$-x_{22}$	$-x_{23}$	1	
1	1	3	6	4	7	0	$= x_0$
1	1	1	0	0	0	7	$= x_{s1}$
0	0	0	1	1	1	5	$= x_{s2}$
-3	0	0	-4	0	0	-16*	$= x_{s3}$
0	-3	0	0	-4	0	- 6	$= x_{s4}$
0	0	-3	0	0	-4	-10	$= x_{s5}$
-1	0	0	0	0	0	0	$= x_{11}$
0	-1	0	0	0	0	0	$= x_{12}$
0	0	-1	0	0	0	0	$= x_{13}$
0	0	0	-1	0	0	0	$= x_{21}$
0	0	0	0	-1	0	0	$= x_{22}$
0	0	0	0	0	-1	0	$= x_{23}$
-1*	0	0	-2	0	0	- 6	$= s_1$

$\lambda = 3$

(1)

$-s_1$	$-x_{12}$	$-x_{13}$	$-x_{21}$	$-x_{22}$	$-x_{23}$	1	
1	1	3	4	4	7	- 6	$= x_0$
1	1	1	-2	0	0	1	$= x_{s1}$
0	0	0	1	1	1	5	$= x_{s2}$
-3	0	0	2	0	0	2	$= x_{s3}$
0	-3	0	0	-4	0	- 6	$= x_{s4}$
0	0	-3	0	0	-4	-10*	$= x_{s5}$
-1	0	0	2	0	0	6	$= x_{11}$
0	-1	0	0	0	0	0	$= x_{12}$
0	0	-1	0	0	0	0	$= x_{13}$
0	0	0	-1	0	0	0	$= x_{21}$
0	0	0	0	-1	0	0	$= x_{22}$
0	0	0	0	0	-1	0	$= x_{23}$
0	0	-1*	0	0	-2	- 4	$= s_2$

$\lambda = 3$

(2)

$-s_1$	$-x_{12}$	$-s_2$	$-x_{21}$	$-x_{22}$	$-x_{23}$	1	
1	1	3	4	4	1	-18	$= x_0$
1	1	1	-2	0	-2	- 3	$= x_{s1}$
0	0	0	1	1	1	5	$= x_{s2}$
-3	0	0	2	0	0	2	$= x_{s3}$
0	-3	0	0	-4	0	-6*	$= x_{s4}$
0	0	-3	0	0	2	2	$= x_{s5}$
-1	0	0	2	0	0	6	$= x_{11}$
0	-1	0	0	0	0	0	$= x_{12}$
0	0	-1	0	0	2	4	$= x_{13}$
0	0	0	-1	0	0	0	$= x_{21}$
0	0	0	0	-1	0	0	$= x_{22}$
0	0	0	0	0	-1	0	$= x_{23}$
0	-1*	0	0	-2	0	-2	$= s_3$

$\lambda = 3$

(3)

$-s_1$	$-s_3$	$-s_2$	$-x_{21}$	$-x_{22}$	$-x_{23}$	1	
1	1	3	4	2	1	-20	$= x_0$
1	1	1	-2	-2	-2	- 5*	$= x_{s1}$
0	0	0	1	1	1	5	$= x_{s2}$
-3	0	0	2	0	0	2	$= x_{s3}$
0	-3	0	0	2	0	0	$= x_{s4}$
0	0	-3	0	0	2	2	$= x_{s5}$
-1	0	0	2	0	0	6	$= x_{11}$
0	-1	0	0	2	0	2	$= x_{12}$
0	0	-1	0	0	2	4	$= x_{13}$
0	0	0	-1	0	0	0	$= x_{21}$
0	0	0	0	-1	0	0	$= x_{22}$
0	0	0	0	0	-1	0	$= x_{23}$
0	0	0	-1	-1	-1*	-3	$= s_4$

$\lambda = 2$

(4)

$-s_1$	$-s_3$	$-s_2$	$-x_{21}$	$-x_{22}$	$-s_4$	1	
1	1	3	3	1	1	-23	$= x_0$
1	1	1	0	0	-2	1	$= x_{s1}$
0	0	0	0	0	1	2	$= x_{s2}$
-3	0	0	2	0	0	2	$= x_{s3}$
0	-3	0	0	2	0	0	$= x_{s4}$
0	0	-3	-2	-2	2	-4*	$= x_{s5}$
-1	0	0	2	0	0	6	$= x_{11}$
0	-1	0	0	2	0	2	$= x_{12}$
0	0	-1	-2	-2	2	-2	$= x_{13}$
0	0	0	-1	0	0	0	$= x_{21}$
0	0	0	0	-1	0	0	$= x_{22}$
0	0	0	1	1	-1	3	$= x_{23}$
0	0	-2	-1	-1*	1	-2	$= s_5$

$\lambda = 2$

(5)

$-s_1$	$-s_3$	$-s_2$	$-x_{21}$	$-s_5$	$-s_4$	1	
1	1	1	2	1	2	-25	$= x_0$
1	1	1	0	0	-2	1	$= x_{s1}$
0	0	0	0	0	1	2	$= x_{s2}$
-3	0	0	2	0	0	2	$= x_{s3}$
0	-3	-4	-2	2	2	- 4*	$= x_{s4}$
0	0	1	0	-2	0	0	$= x_{s5}$
-1	0	0	2	0	0	6	$= x_{11}$
0	-1	-4	-2	2	2	-2	$= x_{12}$
0	0	3	0	-2	0	2	$= x_{13}$
0	0	0	-1	0	0	0	$= x_{21}$
0	0	2	1	-1	-1	2	$= x_{22}$
0	0	-2	0	1	0	1	$= x_{23}$
0	-1	-1*	-1	0	0	-1	$= s_6$

$\lambda = 4$

(6)

$-s_1$	$-s_3$	$-s_6$	$-x_{21}$	$-s_5$	$-s_4$	1	
1	0	1	1	1	2	-26	$= x_0$
1	0	1	-1	0	-2	0	$= x_{s1}$
0	0	0	0	0	1	2	$= x_{s2}$
-3	0	0	2	0	0	2	$= x_{s3}$
0	-1	-4	2	2	2	0	$= x_{s4}$
0	-1	1	-1	-2	0	-1	$= x_{s5}$
-1	0	0	2	0	0	6	$= x_{11}$
0	3	-4	2	2	2	2	$= x_{12}$
0	-3	3	-3	-2	0	-1*	$= x_{13}$
0	0	0	-1	0	0	0	$= x_{21}$
0	-2	2	-1	-1	1	0	$= x_{22}$
0	2	-2	2	1	0	3	$= x_{23}$
0	-1*	1	-1	-1	0	-1	$= s_7$

$\lambda = 3$

(7)

$-s_1$	$-s_7$	$-s_6$	$-x_{21}$	$-s_5$	$-s_4$	1	
1	0	1	1	1	2	-26	$= x_0$
1	0	1	-1	0	-2	0	$= x_{s1}$
0	0	0	0	0	1	2	$= x_{s2}$
-3	0	0	2	0	0	2	$= x_{s3}$
0	1	-3	1	1	2	-1	$= x_{s4}$
0	-1	0	0	-1	0	0	$= x_{s5}$
-1	0	0	2	0	0	6	$= x_{11}$
0	3	-1	-1	-1	2	-1*	$= x_{12}$
0	-3	0	0	1	0	2	$= x_{13}$
0	0	0	-1	0	0	0	$= x_{21}$
0	-2	0	1	1	-1	2	$= x_{22}$
0	2	0	0	-1	0	1	$= x_{23}$
0	0	-1	-1*	-1	2	-1	$= s_8$

$\lambda = 1$

(8)

$-s_1$	$-s_7$	$-s_6$	$-s_8$	$-s_5$	$-s_4$	1	
1	0	0	1	0	4	-27	$= x_0$
1	0	2	-1	1	-4	1	$= x_{s1}$
0	0	0	0	0	1	2	$= x_{s2}$
-3	0	-2	2	-2	4	0	$= x_{s3}$
0	1	-4	1	0	4	-2*	$= x_{s4}$
0	-1	0	0	-1	0	0	$= x_{s5}$
-1	0	-2	2	-2	4	4	$= x_{11}$
0	3	0	-1	0	0	0	$= x_{12}$
0	-3	0	0	1	0	2	$= x_{13}$
0	0	1	-1	1	-2	1	$= x_{21}$
0	-2	-1	1	0	1	1	$= x_{22}$
0	2	0	0	-1	0	1	$= x_{23}$
0	0	-1*	0	0	1	-1	$= s_9$

$\lambda = 4$

(9)

$-s_1$	$-s_7$	$-s_9$	$-s_8$	$-s_5$	$-s_4$	1	
1	0	0	1	0	4	-27	$= x_0$
1	0	2	-1	1	-2	- 1*	$= x_{s1}$
0	0	0	0	0	1	2	$= x_{s2}$
-3	0	-2	2	-2	2	2	$= x_{s3}$
0	1	-4	1	0	0	2	$= x_{s4}$
0	-1	0	0	-1	0	0	$= x_{s5}$
-1	0	-2	2	-2	2	6	$= x_{11}$
0	3	0	-1	0	3	0	$= x_{12}$
0	-3	0	0	1	-3	2	$= x_{13}$
0	0	1	-1	1	1	0	$= x_{21}$
0	-2	-1	1	0	0	2	$= x_{22}$
0	2	0	0	-1	0	1	$= x_{23}$
1	0	2	-1*	1	-2	-1	$= s_{10}$

$\lambda = 1$

(10)

$-s_1$	$-s_7$	$-s_9$	$-s_{10}$	$-s_5$	$-s_4$	1	
2	0	2	1	2	2	-28	$= x_0$
0	0	0	-1	0	-4	0	$= x_{s1}$
0	0	0	0	0	1	2	$= x_{s2}$
-1	0	2	2	0	-2	0	$= x_{s3}$
1	1	-2	1	1	-2	1	$= x_{s4}$
0	-1	0	0	-1	0	0	$= x_{s5}$
1	0	2	2	0	-2	4	$= x_{11}$
-1	3	-2	-1	-1	5	1	$= x_{12}$
0	-3	0	0	1	-3	2	$= x_{13}$
-1	0	-1	-1	0	3	1	$= x_{21}$
1	-2	-1	1	1	2	1	$= x_{22}$
0	2	0	0	-1	0	1	$= x_{23}$

Rule 2 - Rank all the columns in order of descending size. Each eligible row is then assigned a rank equal to the largest rank of its eligible columns. The eligible row with the smallest rank is then selected. In the event that two or more rows have the same smallest rank, the row with the most negative right hand side is selected. (Iterations 4 and 5 of the solution which follows illustrate this). In the event that the right-hand sides are also equal, the last such row in the tableau is selected. (Iteration 6 illustrates this.)

(0)

CR / RR	6 $-X_{11}$	5 $-X_{12}$	4 $-X_{13}$	2 $-X_{21}$	3 $-X_{22}$	1 $-X_{23}$	1	
	1	1	3	6	4	7	0	
	1	1	1	0	0	0	7	
	0	0	0	1	1	1	5	
6	-3	0	0	-4	0	0	-16	
5	0	-3	0	0	-4	0	- 6	
4	0	0	-3	0	0	-4	-10*	
	-1	0	0	0	0	0	0	
	0	-1	0	0	0	0	0	
	0	0	-1	0	0	0	0	
	0	0	0	-1	0	0	0	
	0	0	0	0	-1	0	0	
	0	0	0	0	0	-1	0	
	0	0	-1*	0	0	-2	-4 = S_1	

$\lambda = 3$

CR = Column Rank; RR = Row Rank

RULE 2

(1)

CR / RR	5 $-X_{11}$	4 $-X_{12}$	3 $-S_1$	1 $-X_{21}$	2 $-X_{22}$	6 $-X_{23}$	1		
	1	1	3	6	4	1	-12	=	X_0
	1	1	1	0	0	-2	3	=	X_{S1}
	0	0	0	1	1	1	5	=	X_{S2}
5	-3	0	0	-4	0	0	-16	=	X_{S3}
4	0	-3	0	0	-4	0	- 6*	=	X_{S4}
	0	0	-3	0	0	2	2	=	X_{S5}
	-1	0	0	0	0	0	0	=	X_{11}
	0	-1	0	0	0	0	0	=	X_{12}
	0	0	-1	0	0	2	4	=	X_{13}
	0	0	0	-1	0	0	0	=	X_{21}
	0	0	0	0	-1	0	0	=	X_{22}
	0	0	0	0	0	-1	2	=	X_{23}
	0	-1*	0	0	-2	0	-2	=	S_2

$\lambda = 3$

CR \ RR	5	4	(2) 2	1	3	6	
	$-X_{11}$	$-S_2$	$-S_1$	$-X_{21}$	$-X_{22}$	$-X_{23}$	1
	1	1	3	6	2	1	-14
	1	1	1	0	-2	-2	1
	0	0	0	1	1	1	5
5	-3	0	0	-4	0	0	-16*
	0	-3	0	0	2	0	0
	0	0	-3	0	0	2	2
	-1	0	0	0	0	0	0
	0	-1	0	0	2	0	2
	0	0	-1	0	0	2	4
	0	0	0	-1	0	0	0
	0	0	0	0	-1	0	0
	0	0	0	0	0	-1	0
	-1*	0	0	-2	0	0	-6 = S_3

$\lambda = 3$

CR \ RR	5	4	(3) 2	1	3	6		
	$-S_3$	$-S_2$	$-S_1$	$-X_{21}$	$-X_{22}$	$-X_{23}$	1	
	1	1	3	4	2	1	-20	= X_0
6	1	1	1	-2	-2	-2	- 5*	= X_{S1}
	0	0	0	1	1	1	5	= X_{S2}
	-3	0	0	2	0	0	2	= X_{S3}
	0	-3	0	0	2	0	0	= X_{S4}
	0	0	-3	0	0	2	2	= X_{S5}
	-1	0	0	2	0	0	6	= X_{11}
	0	-1	0	0	2	0	2	= X_{12}
	0	0	-1	0	0	2	4	= X_{13}
	0	0	0	-1	0	0	0	= X_{21}
	0	0	0	0	-1	0	0	= X_{22}
	0	0	0	0	0	-1	0	= X_{23}
	0	0	0	-1	-1	-1*	-3	= S_4

$\lambda = 2$

(4)

CR / RR	4 $-S_3$	3 $-S_2$	1 $-S_1$	2 $-X_{21}$	5 $-X_{22}$	6 $-S_4$	1	
	1	1	3	3	1	1	-23	
	1	1	1	0	0	-2	1	
	0	0	0	0	0	1	2	
	-3	0	0	2	0	0	2	
	0	-3	0	0	2	0	0	
5	0	0	-3	-2	-2	2	-4*	
	-1	0	0	2	0	0	6	
	0	-1	0	0	2	0	2	
5	0	0	-1	-2	-2	2	-2	
	0	0	0	-1	0	0	0	
	0	0	0	0	-1	0	0	
	0	0	0	1	1	-1	3	
	0	0	-2	-1	-1*	1	-2	$= S_5$

$\lambda = 2$

(5)

CR / RR	5 $-S_3$	3 $-S_2$	4 $-S_1$	1 $-X_{21}$	6 $-S_5$	2 $-S_4$	1		
	1	1	1	2	1	2	-25	=	X_0
	1	1	1	0	0	-2	1	=	X_{S1}
	0	0	0	0	0	1	2	=	X_{S2}
	-3	0	0	2	0	0	2	=	X_{S3}
4	0	-3	-4	-2	2	2	-4*	=	X_{S4}
	0	0	1	0	-2	0	0	=	X_{S5}
	-1	0	0	2	0	0	6	=	X_{11}
4	0	-1	-4	-2	2	2	-2	=	X_{12}
	0	0	3	0	-2	0	2	=	X_{13}
	0	0	0	-1	0	0	0	=	X_{21}
	0	0	2	1	-1	-1	2	=	X_{22}
	0	0	-2	0	1	0	1	=	X_{23}
	0	-1	-1*	-1	0	0	-1	=	S_6

$\lambda = 4$

(6)

CR / RR	3 $-S_3$	6 $-S_2$	2 $-S_6$	5 $-X_{21}$	4 $-S_5$	1 $-S_4$	1	
	1	0	1	1	1	2	-26	
	1	0	1	-1	0	-2	0	
	0	0	0	0	0	1	2	
	-3	0	0	2	0	0	2	
	0	1	-4	2	2	2	0	
6	0	-1	1	-1	-2	0	-1	
	-1	0	0	2	0	0	6	
	0	3	-4	2	2	2	2	
6	0	-3	3	-3	-2	0	-1*	
	0	0	0	-1	0	0	0	
	0	-2	2	-1	-1	-1	0	
	0	2	-2	2	1	0	3	
	0	-1*	1	-1	-1	0	-1	$= S_7$

$\lambda = 3$

(7)

CR / RR	3 $-S_3$	6 $-S_7$	2 $-S_6$	5 $-X_{21}$	4 $-S_5$	1 $-S_4$	1		
	1	0	1	1	1	2	-26	=	X_0
	1	0	1	-1	0	-2	0	=	X_{S1}
	0	0	0	0	0	1	2	=	X_{S2}
	-3	0	0	2	0	0	2	=	X_{S3}
2	0	1	-3	1	1	2	-1*	=	X_{S4}
	0	-1	0	0	1	0	0	=	X_{S5}
	-1	0	0	2	0	0	6	=	X_{11}
5	0	3	-1	-1	-1	2	-1	=	X_{12}
	0	-3	0	0	1	0	2	=	X_{13}
	0	0	0	-1	0	0	0	=	X_{21}
	0	-2	0	1	1	-1	2	=	X_{22}
	0	2	0	0	-1	0	1	=	X_{23}
	0	0	-1*	0	0	0	-1	=	S_8

$\lambda = 3$

(8)

CR / RR	3 $-S_3$	6 $-S_7$	2 $-S_8$	5 $-X_{21}$	4 $-S_5$	1 $-S_4$	1
	1	0	1	1	1	2	-27
5	1	0	1	-1	0	-2	- 1*
	0	0	0	0	0	1	2
	-3	0	0	2	0	0	2
	0	1	-3	1	1	2	2
	0	-1	0	0	1	0	0
	0	-1	0	0	1	0	0
	-1	0	0	2	0	0	6
	0	-3	-1	-1	-1	2	0
	0	-3	0	0	1	0	2
	0	0	0	-1	0	0	0
	0	-2	0	1	1	-1	2
	0	2	0	0	-1	0	1
	0	0	-1*	0	0	-2	-1 = S_9

$\lambda = 1$

(9)

CR / RR	$-S_3$	$-S_7$	$-S_8$	$-S_9$	$-S_5$	$-S_4$	1			
	1	0	1	1	1	0	-28	=	X_0	
	1	0	1	1	0	0	0	=	X_{S1}	
	0	0	0	0	0	1	2	=	X_{S2}	
	-3	0	0	0	0	-4	0	=	X_{S3}	
	0	1	-3	-3	1	0	1	=	X_{S4}	
	0	-1	0	0	1	0	0	=	X_{S5}	∞
	-1	0	0	0	0	-4	4	=	X_{11}	
	0	-3	-1	-1	-1	0	1	=	X_{12}	
	0	-3	0	0	1	0	2	=	X_{13}	
	0	0	0	0	0	2	1	=	X_{21}	
	0	-2	0	0	1	-3	1	=	X_{22}	
	0	2	0	0	-1	0	1	=	X_{23}	

Rule 3 - Rank each eligible row according to the number of zeros encountered at the top of its smallest eligible column. (Thus, a rank of zero means there are no zeroes at the top of the smallest eligible column.) From the eligible rows, the row having the lowest rank is selected.

If more than one row has an equally low rank and the rank is zero, the row whose pivot column has the largest value at the top is selected. (Iterations 0 and 3 of the solution which follows illustrate this.) If these values are also equal, the highest such row in the tableau is selected. (Iterations 1 and 7 illustrate this.)

If, as above, more than one row has an equally low rank and the rank is greater than zero the highest such row in the tableau is selected.

RULE 3

(0)

CR	0	0	0	0	0	0	
RR	$-x_{11}$	$-x_{12}$	$-x_{13}$	$-x_{21}$	$-x_{22}$	$-x_{23}$	1
	1	1	3	6	4	7	0
	1	1	1	0	0	0	7
	0	0	0	1	1	1	5
0	-3	0	0	-4	0	0	-16
0	0	-3	0	0	-4	0	-6
0	0	0	-3	0	0	-4	-10*
	-1	0	0	0	0	0	0
	0	-1	0	0	0	0	0
	0	0	-1	0	0	0	0
	0	0	0	-1	0	0	0
	0	0	0	0	-1	0	0
	0	0	0	0	0	-1	0
	0	0	-1*	0	0	-2	-4 = s_1

$\lambda = 3$

CR = Column Rank; RR = Row Rank

(1)

CR	0	0	0	0	0	0		
RR	$-x_{11}$	$-x_{12}$	$-s_1$	$-x_{21}$	$-x_{22}$	$-x_{23}$	1	
	1	1	3	6	4	1	-12	= x_0
	1	1	1	0	0	-2	3	= x_{s1}
	0	0	0	1	1	1	5	= x_{s2}
0	-3	0	0	-4	0	0	-16*	= x_{s3}
0	0	-3	0	0	-4	0	-6	= x_{s4}
	0	0	-3	0	0	2	2	= x_{s5}
	-1	0	0	0	0	0	0	= x_{11}
	0	-1	0	0	0	0	0	= x_{12}
	0	0	-1	0	0	2	4	= x_{13}
	0	0	0	-1	0	0	0	= x_{21}
	0	0	0	0	-1	0	0	= x_{22}
	0	0	0	0	0	-1	0	= x_{23}
-1*	0	0	-2	0	0	-6	= s_2	

$\lambda = 3$

(2)

CR / RR	0 $-s_2$	0 $-x_{12}$	0 $-s_1$	0 $-x_{21}$	0 $-x_{22}$	0 $-x_{23}$	1	
	1	1	3	4	4	1	-18	
0	1	1	1	-2	0	-2	-3*	
	0	0	0	1	1	1	5	
	-3	0	0	2	0	0	2	
0	0	-3	0	0	-4	0	-6	
	0	0	-3	0	0	2	2	
	0	0	-3	0	0	2	2	
	-1	0	0	2	0	0	6	
	0	-1	0	0	0	0	0	
	0	0	-1	0	0	2	4	
	0	0	0	-1	0	0	0	
	0	0	0	0	-1	0	0	
	0	0	0	0	0	-1	0	
	0	0	0	-1	-2	=	s_3	

$\lambda = 2$

(3)

CR / RR	0 $-s_2$	0 $-x_{12}$	0 $-s_1$	0 $-x_{21}$	0 $-x_{22}$	0 $-s_3$	1		
	1	1	3	3	4	1	-20	=	x_0
	1	1	1	0	0	-2	1	=	x_{s1}
	0	0	0	0	1	1	3	=	x_{s2}
	-3	0	0	2	0	0	2	=	x_{s3}
0	0	-3	0	0	-4	0	-6	=	x_{s4}
0	0	0	-3	-2	0	2	-2*	=	x_{s5}
	-1	0	0	2	0	0	6	=	x_{11}
	0	-1	0	0	0	0	0	=	x_{12}
	0	0	-1	-2	0	2	0	=	x_{13}
	0	0	0	-1	0	0	0	=	x_{21}
	0	0	0	0	-1	0	0	=	x_{22}
	0	0	0	1	0	-1	2	=	x_{23}
	0	0	-1	-1*	0	0	-1	=	s_4

$\lambda = 3$

(4)

CR / RR	0 $-s_2$	0 $-x_{12}$	1 $-s_1$	0 $-s_4$	0 $-x_{22}$	0 $-s_3$	1	
	1	1	0	3	4	1	-23	
	1	1	1	0	0	-2	1	
	0	0	0	0	1	1	3	
	-3	0	-2	2	0	0	0	
0	0	-3	0	0	-4	0	-6*	
	0	0	-1	-2	0	2	0	
	-1	0	-2	2	0	0	4	
	0	-1	0	0	0	0	0	
	0	0	1	-2	0	2	2	
	0	0	1	-1	0	0	1	
	0	0	0	0	-1	0	0	
	0	0	-1	1	0	-1	1	
	0	-1*	0	0	-2	0	-2	= s_5

$\lambda = 3$

(5)

CR / RR	0 $-s_2$	0 $-s_5$	1 $-s_1$	0 $-s_4$	0 $-x_{22}$	0 $-s_3$	1		
	1	1	0	3	2	1	-25	=	x_0
0	1	1	1	0	-2	-2	-1	=	x_{s1}
	0	0	0	0	1	1	3	=	x_{s2}
	-3	0	-2	2	0	0	0	=	x_{s3}
	0	-3	0	0	2	0	0	=	x_{s4}
	0	0	-1	-2	0	2	0	=	x_{s5}
	-1	0	-2	2	0	0	4	=	x_{11}
	0	-1	0	0	2	0	2	=	x_{12}
	0	0	1	-2	0	2	2	=	x_{13}
	0	0	1	-1	0	0	1	=	x_{21}
	0	0	0	0	-1	0	0	=	x_{22}
	0	0	-1	1	0	-1	1	=	x_{23}
	0	0	0	0	-1	-1*	-1	=	s_6

$\lambda = 3$

(6)

CR	0	0	1	0	0	0		
RR	$-s_2$	$-s_5$	$-s_1$	$-s_4$	$-x_{22}$	$-s_6$	1	
	1	1	0	3	1	1	-26	
	1	1	1	0	0	-2	1	
	0	0	0	0	0	1	2	
	-3	0	-2	2	0	0	0	
	0	-3	0	0	2	0	0	
1	0	0	-1	-2	-2	2	-2*	
	-1	0	-2	2	0	0	4	
	0	-1	0	0	2	0	2	
	0	0	1	-2	-2	2	0	
	0	0	1	-1	0	0	1	
	0	0	0	0	-1	0	0	
	0	0	-1	1	1	-1	2	
	0	0	-1*	-2	-2	2	-2	= s_7

$\lambda = 1$

(7)

CR	0	0	1	0	0	0		
RR	$-s_2$	$-s_5$	$-s_7$	$-s_4$	$-x_{22}$	$-s_6$	1	
	1	1	0	3	1	1	-26	= x_0
0	1	1	1	-2	-2	0	-1*	= x_{s1}
	0	0	0	0	0	1	2	= x_{s2}
	-3	0	-2	6	4	-4	4	= x_{s3}
	0	-3	0	0	2	0	0	= x_{s4}
	0	0	-1	0	0	0	0	= x_{s5}
	-1	0	-2	6	4	-4	8	= x_{11}
	0	-1	0	0	2	0	2	= x_{12}
0	0	0	1	-4	-4	4	-2	= x_{13}
0	0	0	1	-3	-2	2	-1	= x_{21}
	0	0	0	0	-1	0	0	= x_{22}
	0	0	-1	3	3	-3	4	= x_{23}
	0	0	0	-1	-1*	0	-1	= s_8

$\lambda = 2$

(8)

CR	0	0	1	0	0	0	
RR	$-s_2$	$-s_5$	$-s_7$	$-s_4$	$-s_8$	$-s_6$	1
	1	1	0	2	1	1	-27
	1	1	1	0	-2	0	1
	0	0	0	0	0	1	2
	-3	0	-2	2	4	-4	0
0	0	-3	0	-2	2	0	-2*
	0	0	-1	0	0	0	0
	-1	0	-2	2	4	-4	4
	0	-1	0	-2	2	0	0
	0	0	1	0	-4	4	2
	0	0	1	-1	-2	2	1
	0	0	0	1	-1	0	1
	0	0	-1	0	3	-3	1
	0	-1*	0	-1	0	0	-1 = s_9

$\lambda = 3$

(9)

CR								
RR	$-s_2$	$-s_9$	$-s_7$	$-s_4$	$-s_8$	$-s_6$	1	
	1	1	0	1	1	1	28	= x_0
	1	1	1	-1	-2	0	0	= x_{s1}
	0	0	0	0	0	1	2	= x_{s2}
	-3	0	-2	2	4	-4	0	= x_{s3}
	0	-3	0	1	2	0	1	= x_{s4}
	0	0	-1	0	0	0	0	= x_{s5}
	-1	0	-2	2	4	-4	4	= x_{11}
	0	-1	0	-1	2	0	1	= x_{12}
	0	0	1	0	-4	4	2	= x_{13}
	0	0	1	-1	-2	2	1	= x_{21}
	0	0	0	1	-1	0	1	= x_{22}
	0	0	-1	0	3	-3	1	= x_{23}

APPENDIX B

The 25 Test Problems

Table B-1 below provides the ranges of values assigned to the constants (c_{ij}, a_i, b_j, p_i) of the test problems. The actual values used for the problems were selected for within these ranges through the use of a random number table[5]. Table B-2 gives the actual values used in the problems as well as the solution to each problem that was solved. In order to conserve space, only the coefficients of each variable are given in Table 2, with the variables appearing at the top of each page.

	Constants			
Problem Number	c_{ij}	a_i	b_j	p_i
1	1-100	1-50	25-75	1-10
2	1-100	1-50	80-200	1-10
3	1-100	1-50	80-200	1-10
4	1-20	1-50	50-100	1-10
5	1-20	1-50	70-100	1-10
6	1-20	10-50	150-200	1-10
7	4-20	100-200	699	2-11
8	4-20	100-200	1000-1100	2-11
9	5	100-200	600-800	2-11
10	5-7	100-200	700-850	2-11
11	1-20	10-50	70-80	1-10
12	5-15	10-50	130-160	2-11
13	4-20	100-200	1010-1060	2-11
14	4-20	85-110	500-1000	2-11
15	4-20	160-190	500-1000	6
16	1-10	200-400	1420-1450	2-11
17	1-10	400-500	1000-1650	2-11
18	10-20	50-100	400-500	2-11
19	10-20	50-100	175-225	2-11
20	5-15	50-60	100-170	2-11
21	50-60	100-200	600-800	2-11
22	80-100	100-200	650-720	2-11
23	50-100	100-200	500-700	2-11
24	4-20	100-200	1600-2000	20-30
25	4-20	100-500	5000-8000	10-50

TABLE B-1

Test Problem Constant Ranges

TABLE B-2
Test Problems

Prob. No.		Variables													
		x_{11}	x_{12}	x_{13}	x_{14}	x_{21}	x_{22}	x_{23}	x_{24}	x_{31}	x_{32}	x_{33}	x_{34}		RHS
1	Obj. Fnc.	10	22	24	42	37	77	99	96	89	85	28	63		
	Constr.	1	1	1	1									≤	9
						1	1	1	1					≤	8
										1	1	1	1	≤	7
		10				5				9				≥	60
			10				5				9			≥	55
				10				5				9		≥	45
					10				5				9	≥	33
	Sol.	3	5	0	1	6	1	0	1	0	0	5	2	=	843
2	Obj Fnc.	15	46	48	93	39	6	72	91	14	36	69	40		
	Constr.	1	1	1	1									≤	12
						1	1	1	1					≤	21
										1	1	1	1	≤	31
		2				10				6				≥	91
			2				10				6			≥	104
				2				10				6		≥	113
					2				10				6	≥	88
	Sol.	1	0	2	0	0	10	11	0	15	1	0	15	=	1809
3	Obj. Fnc.	1	25	22	6	81	11	56	5	63	3	88	48		
	Constr.	1	1	1	1									≤	33
						1	1	1	1					≤	29
										1	1	1	1	≤	33
		6				8				6				≥	66
			6				8				6			≥	95
				6				8				6		≥	88
					6				8				6	≥	92
	Sol.	11	0	15	0	0	0	0	12	0	16	0	0	=	449

Prob. No.		Variables													
		x_{11}	x_{12}	x_{13}	x_{14}	x_{21}	x_{22}	x_{23}	x_{24}	x_{31}	x_{32}	x_{33}	x_{34}		RHS
4	Obj. Fnc.	2	16	7	10	3	8	9	14	20	7	12	7		
	Constr.	1	1	1	1									≤	29
						1	1	1	1					≤	31
		1				5				1	1	1	1	≤	25
			1				5			7				≥	62
				1				5			7			≥	93
					1				5			7		≥	71
													7	≥	90
	Sol.	0	0	1	0	13	2	14	0	0	12	0	13	=	363
5	Obj. Fnc.	16	20	18	13	16	19	4	14	6	20	18	17		
	Constr.	1	1	1	1									≤	25
						1	1	1	1					≤	38
		3				6				1	1	1	1	≤	24
			3				6			2				≥	90
				3				6			2			≥	86
					3				6			2		≥	81
													2	≥	72
	Sol.	0	1	0	18	7	14	14	3	24	0	0	0	=	874
6	Obj. Fnc.	14	18	5	17	13	4	15	14	8	20	8	1		
	Constr.	1	1	1	1									≤	47
						1	1	1	1					≤	12
		7				8				1	1	1	1	≤	23
			7				8			8				≥	150
				7				8			8			≥	125
					7				8			8		≥	182
													8	≥	152
	Sol.	18	3	26	0	0	12	0	0	3	1	0	19	=	547

Prob. No.		Variables													
		x_{11}	x_{12}	x_{13}	x_{14}	x_{21}	x_{22}	x_{23}	x_{24}	x_{31}	x_{32}	x_{33}	x_{34}		RHS
7	Obj. Fnc.	16	18	16	9	16	11	18	8	9	16	20	19		
	Constr.	1	1	1	1									≤	124
						1	1	1	1					≤	143
		2				8				1	1	1	1	≤	156
			2				8			9				≥	699
				2				8			9			≥	699
					2				8			9		≥	699
													9	≥	699
	Sol.	Unsolved													
8	Obj. Fnc.	4	5	14	17	20	6	7	5	4	5	14	10		
	Constr.	1	1	1	1									≤	142
						1	1	1	1					≤	199
		3				11				10				≥	1048
			3				11				10			≥	1036
				3				11				10		≥	1049
					3				11				10	≥	1100
	Sol.	136	4	2	0	0	4	95	100	64	98	0	0	=	2527
9	Obj. Fnc.	5	5	5	5	5	5	5	5	5	5	5	5		
	Constr.	1	1	1	1									≤	144
						1	1	1	1					≤	101
		5				9				1	1	1	1	≤	142
			5				9			8				≥	676
				5				9			8			≥	670
					5				9			8		≥	622
													8	≥	716
	Sol.	126	1	0	0	2	73	22	4	3	1	53	85	=	1850

Prob. No.		Variables													
		x_{11}	x_{12}	x_{13}	x_{14}	x_{21}	x_{22}	x_{23}	x_{24}	x_{31}	x_{32}	x_{33}	x_{34}		RHS
10	Obj. Fnc.	7	6	6	7	5	6	5	6	6	6	6	5		
	Constr.	1	1	1	1									≤	130
						1	1	1	1					≤	173
										1	1	1	1	≤	156
		4				10				6				≥	781
							10				6			≥	760
			4					10				6		≥	761
				4					10				6	≥	791
	Sol.	0	107	1	0	77	20	76	0	2	22	0	132	=	2337
11	Obj. Fnc.	13	12	17	15	4	5	9	8	16	15	11	16		
	Constr.	1	1	1	1									≤	12
						1	1	1	1					≤	27
										1	1	1	1	≤	14
		7				7				3				≥	74
			7				7				3			≥	79
				7				7				3		≥	71
					7				7				-3	≥	79
	Sol.	0	11	0	1	11	0	6	10	0	1	10	1	=	466
12	Obj. Fnc.	15	14	6	5	9	12	6	6	5	11	7	5		
	Constr.	1	1	1	1									≤	41
						1	1	1	1					≤	15
										1	1	1	1	≤	20
		9				8				6				≥	151
			9				8				6			≥	149
				9				8				6		≥	159
					9				8				6	≥	146
	Sol.	0	13	15	13	4	4	3	4	20	0	0	0	=	563

Prob. No.		Variables												
		x_{11}	x_{12}	x_{13}	x_{14}	x_{21}	x_{22}	x_{23}	x_{24}	x_{31}	x_{32}	x_{33}	x_{34}	RHS
13	Obj. Fnc.	16	16	5	12	14	13	19	16	6	7	19	5	
	Constr.	1	1	1	1									≤ 192
						1	1	1	1					≤ 104
										1	1	1	1	≤ 150
		10				10				8				≥ 1014
			10				10				8			≥ 1036
				10				10				8		≥ 1020
					10				10				8	≥ 1022
	Sol.	0	0	102	84	0	104	0	0	127	0	0	23	= 3747
14	Obj. Fnc.	13	10	19	11	17	11	11	6	16	6	18	14	
	Constr.	1	1	1	1									≤ 110
						1	1	1	1					≤ 96
										1	1	1	1	≤ 106
		11				7				11				≥ 527
			11				7				11			≥ 669
				11				7				11		≥ 947
					11				7				11	≥ 655
	Sol.	48	0	1	40	0	0	63	31	0	61	45	0	= 3138
15	Obj. Fnc.	17	12	11	11	4	20	9	15	11	20	14	11	
	Constr.	1	1	1	1									≤ 174
						1	1	1	1					≤ 187
										1	1	1	1	≤ 167
		6				6				6				≥ 873
			6				6				6			≥ 724
				6				6				6		≥ 998
					6				6				6	≥ 558
	Sol.	0	121	53	0	146	0	41	0	0	0	73	93	= 5033

Prob. No.		Variables													
		x_{11}	x_{12}	x_{13}	x_{14}	x_{21}	x_{22}	x_{23}	x_{24}	x_{31}	x_{32}	x_{33}	x_{34}		RHS
16	Obj. Fnc.	2	3	10	2	10	5	10	7	9	4	2	10		
		1	1	1	1									≤	304
						1	1	1	1					≤	273
										1	1	1	1	≤	241
	Constr.	9				4				8				≥	1439
			9				4				8			≥	1438
				9				4				8		≥	1444
					9				4				8	≥	1431
	Sol.	160	0	0	144	0	238	1	34	0	61	180	0	=	2650
17	Obj. Fnc.	10	1	3	5	3	3	9	3	1	8	4	4		
		1	1	1	1									≤	449
						1	1	1	1					≤	449
										1	1	1	1	≤	454
	Constr.	3				7				2				≥	1360
			3				7				2			≥	1390
				3				7				2		≥	1003
					3				7				2	≥	1580
	Sol.	0	116	332	0	72	149	1	226	428	0	0	0	=	2890
18	Obj. Fnc.	12	11	13	12	14	12	12	17	19	20	20	13		
		1	1	1	1									≤	69
						1	1	1	1					≤	56
										1	1	1	1	≤	77
	Constr.	10				8				9				≥	441
			10				8				9			≥	412
				10				8				9		≥	482
					10				8				9	≥	494
	Sol.	28	38	3	0	0	4	52	0	18	0	4	55	=	2602

Prob. No.		Variables													
		x_{11}	x_{12}	x_{13}	x_{14}	x_{21}	x_{22}	x_{23}	x_{24}	x_{31}	x_{32}	x_{33}	x_{34}		RHS
19	Obj. Fnc.	13	13	10	13	11	18	16	19	12	12	12	12		
	Constr.	1	1	1	1									≤	87
						1	1	1	1					≤	71
										1	1	1	1	≤	51
		3				2				8				≥	212
			3				2				8			≥	220
				3				2				8		≥	179
					3				2				8	≥	196
	Sol.	20	4	59	4	68	0	1	0	2	26	0	23	=	2330
20	Obj. Fnc.	11	11	12	6	14	8	7	5	9	15	8	12		
	Constr.	1	1	1	1									≤	57
						1	1	1	1					≤	60
										1	1	1	1	≤	58
		5				3				3				≥	104
			5				3				3			≥	119
				5				3				3		≥	146
					5				3				3	≥	112
	Sol.	19	22	0	16	0	3	46	11	3	0	3	0	=	999
21	Obj. Fnc.	60	51	58	59	56	53	55	59	54	58	58	53		
	Constr.	1	1	1	1									≤	173
						1	1	1	1					≤	182
										1	1	1	1	≤	128
		2				9				7				≥	729
			2				9				7			≥	740
				2				9				7		≥	671
					2				9				7	≥	625
	Sol.	Unsolved													

Prob. No.		Variables												
		x_{11}	x_{12}	x_{13}	x_{14}	x_{21}	x_{22}	x_{23}	x_{24}	x_{31}	x_{32}	x_{33}	x_{34}	RHS
22	Obj. Fnc.	81	96	84	86	88	90	91	92	87	88	96	96	
	Constr.	1	1	1	1									≤ 194
						1	1	1	1					≤ 145
										1	1	1	1	≤ 160
		3				11				5				≥ 707
			3				11				5			≥ 651
				3				11				5		≥ 683
					3				11				5	≥ 663
	Sol.	98	0	4	1	3	21	61	60	76	84	0	0	= 35589
23	Obj. Fnc.	50	75	97	65	70	95	83	62	54	53	87	98	
	Constr.	1	1	1	1									≤ 102
						1	1	1	1					≤ 155
										1	1	1	1	≤ 167
		2				6				8				≥ 657
			2				6				8			≥ 697
				2				6				8		≥ 580
					2				6				8	≥ 511
	Sol.	89	1	0	1	0	0	70	85	60	87	20	0	= 25261
24	Obj. Fnc.	5	9	7	14	13	14	18	10	8	18	11	17	
	Constr.	1	1	1	1									≤ 100
						1	1	1	1					≤ 110
										1	1	1	1	≤ 106
		21				29				26				≥ 1665
			21				29				26			≥ 1924
				21				29				26		≥ 1710
					21				29				26	≥ 1860
	Sol.	Unsolved												

Prob. No. 25

		x_{11}	x_{12}	x_{13}	x_{14}	x_{21}	x_{22}	x_{23}	x_{24}	x_{31}	x_{32}	x_{33}	x_{34}		RHS
Obj. Fnc.		9	20	6	7	5	10	20	5	11	5	20	20		
Constr.		1	1	1	1									≤	374
						1	1	1	1					≤	222
										1	1	1	1	≤	376
		40				28				21				≥	6204
			40				28				21			≥	5716
				40				28				21		≥	7626
					40				28				21	≥	7453
Sol.	Unsolved														

BIBLIOGRAPHY

1. Balas, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables", Operations Research, Vol. 13 (1965),

2. Balinski, M. L., "Integer Programming: Methods, Uses, Computation", Management Science, Vol. 12 (1965), pp. 253-313.

3. Balinski, M. L., "Some General Methods in Integer Programming", Non-Linear Programming, Abadio, J., Editor, North Holland Publishing Company, Amsterdam, 1967.

4. Beale, E. M. L., "Survey of Integer Programming", Operational Research Quarterly, Vol. 16 (1965), pp. 219-228.

5. Beyer, W. H., Editor, Handbook of Tables for Probability and Statistics, the Chemical Rubber Company, Clevelend, Ohio, 1966, pp. 341-345.

6. Dantzig, G. B., Linear Programming and Extensions, Princeton University Press, Princeton, N. J., 1963.

7. Dantzig, G. B., Orden, A., and P. Wolfe, "Generalized Simplex Method for Minimizing a Linear Form Under Linear Inequality Restraints", Rand Report RM-1264, The Rand Corp., Santa Monica, California, 1954.

8. Duncan, D. B., "Multiple Range and Multiple F Tests", Biometrics, Vol. 11 (1955), pp. 1-42.

9. Gelfond, A. O., The Solution of Equations in Integers, P. Noordhoff, Gromingen, The Netherlands, 1960.

10. Glover, F., "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem", Operations Research, Vol. 13 (1965), pp. 879-919.

11. Glover, F., "Stronger Cuts in Integer Programming", Operations Research, Vol. 15 (1967), pp. 1174-1177.

12. Glover, F., "Truncated Enumeration Methods for Solving Pure and Mixed Integer Linear Programs", Working Paper No. 27, Operations Research Center, University of California, Berkeley, May, 1966.

BIBLIOGRAPHY (cont'd)

13. Gomory, R. E., "All-Integer Integer Programming Algirthm", Industrial Scheduling, Muth, J. F., and G. L. Thompson, Editors, Prentice-Hall, New York, 1963, pp. 193-206.

14. Gomory, R. E., "An Algorithm for Integer Solutions to Linear Programs", Recent Advances in Mathematical Programming, Graves, R. L., and P. Wolfe, Editors, McGraw-Hill Book Company, 1963, pp. 269-302.

15. Gomory, R. E., "An Algorithm for the Mixed Integer Problem", Rand Report RM-2597, Rand Corporation, July 7, 1960.

16. Gomory, R. E., "Faces of an Integer Polyhedron", Proceedings of the National Academy of Sciences, Vol. 57 (1967), pp. 16-18.

17. Gomory, R. E., "On the Relation Between Integer and Non-Integer Solutions to Linear Programs", Proceedings of the National Academy of Sciences, Vol. 53 (1965), pp. 260-265.

18. Graves, R. L., and P. Wolfe, Editors, Recent Advances in Mathematical Programming, McGraw-Hill Book Company, Inc., 1963.

19. Haldi, J., and L. M. Isaacson, "A Computer Code for Integer Solutions to Linear Programs", Operations Research, Vol. 13 (1965), pp. 946-958.

20. Hadley, G., Linear Programming, Addison-Wesley, Reading, Massachusetts, 1962.

21. Healy, W. C., Jr., "Multiple Choice Programming", Operations Research, Vol. 12, (1964), pp. 122-138.

22. Land, A. H., and A. G. Doig, "An Automatic Method of Solving Discrete Programming Problems", Econometrica, Vol. 28 (1960), pp. 497-520.

23. Lawler, E. L., and D. E. Woods, "Branch and Bound Methods: A Survey", Operations Research, Vol. 14, (1966), pp. 699-719.

24. Lemke, C. E., "The Dual Method of Solving the Linear Programming Problem", Naval Research Logistics Quarterly, Vol. 1, (1954), pp. 36-47.

25. Lemke, C. E., and K. Speilberg, "Direct Search Zero-One and Mixed Integer Programming", Report #3, IBM Data Processing Division, New York Scientific Center, June, 1966.

BIBLIOGRAPHY (cont'd)

26. Martin, G. T., "An Accelerated Euclidean Algorithm for Integer Linear Programming", Recent Advances in Mathematical Programming, Graves, R. L., and P. Wolfe, Editors, McGraw-Hill Book Company, 1963, pp. 311-318.

27. Peterson, C. C., "Integer Linear Programming", The Journal of Industrial Engineering, August, 1967, pp. 456-464.

28. Schmaltz, J. H., "Allocation of Core Storage for ESS", Technical Report CC2396, Western Electric Company, Engineering Research Center, Princeton, N. J., Nov. 10, 1967.

29. Srinivasan, A. V., "An Investigation of Some Computational Aspects of Integer Programming", Journal of the Association for Computing Machinery, Vol. 12 (1965), pp. 525-535.

30. Story, A. E., and H. M. Wagner, "Computational Experience with Integer Programming for Job-Shop Scheduling", Industrial Scheduling, Muth, J. F., and G. L. Thompson, Editors, Prentice-Hall, New York, 1963.

31. Trauth, C. A., and R. E. Woolsey, Practical Aspects of Integer Linear Programming, Sandia Corporation Monograph SC-R-66-925.

32. White, W., "Comments on a Paper by Bowman", Operations Research, Vol. 9 (1961), pp. 274-276.

33. Wilson, R. B., "Stronger Cuts in Gomory's All-Integer Programming Algorithm", Operations Research, Vol. 15, (1967), pp. 155-156.

34. Woolsey, R. E., and C. A. Trauth, IPSC A Machine Independent Integer Linear Program, Sandia Corporation Monograph SC-RR-66-433, July, 1966.

VITA

PERSONAL HISTORY

Name:	Bruce Allen Weingartner
Birth Place;	East Orange, New Jersey
Birthdate:	February 23, 1940
Parents:	William and Lydia Weingartner
Wife:	Angela Maria (Sirianni) Weingartner
Children:	Kenneth Paul

EDUCATIONAL BACKGROUND

Clarkson College of Technology
Bachelor of Electrical Engineering — 1957 - 1961

Lehigh University
Candidate for Master of Science
in Industrial Engineering — 1966 - 1968

PROFESSIONAL EXPERIENCE

Western Electric Co., Inc.
Winston-Salem, North Carolina
Planning Engineer — 1961 - 1966

Western Electric Co., Inc.
Princeton, New Jersey
Research Engineer — 1966 - 1968